2016

FPGA Realization of Low Register Systolic All One-Polynomial Multipliers Over $GF(2^m)$ and their Applications in Trinomial Multipliers

Pingxiuqi Chen
Wright State University

FPGA REALIZATION OF LOW REGISTER SYSTOLIC ALL ONE-POLYNOMIAL MULTIPLIERS OVER $GF(2^m)$ AND THEIR APPLICATIONS IN TRINOMIAL MULTIPLIERS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering

By

PINGXIUQI CHEN

B.S., Hainan Normal University, China, June 2014

2016

Wright State University

WRIGHT STATE UNIVERSITY

GRADUATE SCHOOL

April 21,2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Pingxiuqi Chen ENTITLED FPGA REALIZATION OF LOW REGISTER SYSTOLIC ALL ONE -POLYNOMIAL MULTIPLIERS OVER $GF(2^m)$ AND THEIR APPLICATIONS IN TRINOMIAL MULTIPLIERS. BE ACCEPT IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Electrical Engineering.

Jiafeng Xie,Ph.D.
Thesis Director

Brian Rigling,Ph.D.
Chair, Electrical Engineering

Committee on Final Examination

Jiafeng Xie,Ph.D.

Henry Chen,Ph.D.

Zhiqiang Wu,Ph.D.

Robert E.W.Fyffe,Ph.D.
Vice President for Research and
Dean of the Graduate School

Abstract

Pingxiuqi,Chen. M.S.E.E., Department of Electrical Engineering, Wright State University, 2016. FPGA realization of low register systolic all one-polynomial multipliers over $GF(2^m)$ and their applications in trinomial multipliers.

All-one-polynomial (AOP)-based systolic multipliers over $GF(2^m)$ are usually not considered for practical implementation of cryptosystems such as elliptic curve cryptography (ECC) due to security reasons. Besides that, systolic AOP multipliers usually suffer from the problem of high register-complexity, especially in field-programmable gate array (FPGA) platforms where the register resources are not that abundant. In this thesis, however, we have shown that the AOP-based systolic multipliers can easily achieve low register-complexity implementations and the proposed architectures can be employed as computation cores to derive efficient implementations of systolic Montgomery multipliers based on trinomials, which are recommended by the National Institute of Standards and Technology (NIST) for cryptosystems. In this paper, first, we propose a novel data broadcasting scheme in which the register-complexity involved within existing AOP-based systolic multipliers is significantly reduced. We have found out that for practical usage, the modified AOP-based systolic structure can be packed as a standard computation core. Next, we propose a novel Montgomery multiplication algorithm that can fully employ the proposed AOP-based computation core. The proposed Montgomery algorithm employs a novel pre-computed-modular (PCM) operation, and the systolic structures based on this algorithm fully inherit the advantages brought from the AOP-based core (low register-complexity, low critical-path delay, and low latency) except some marginal hardware overhead brought by a pre-computation unit. The proposed architectures are then implemented by Xilinx ISE 14.1 and it is shown that compared with the existing designs, the proposed designs achieve at least 70.0% and 47.6% less area-delay product (ADP) and power-delay product (PDP) than the best of competing designs, respectively.

Key Words-Finite field multiplication, systolic structure, low complexity, Montgomery algorithm, irreducible trinomials.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter will give the outline of whole thesis. It presents some basic ideas about related algorithms, and the corresponding structures based on these algorithms. The contributions of this report are also given.

1.1 Preliminary

In recent years, the finite field algorithm as one of the high efficiency and low complexity algorithms has already been used in various fields, such as error-control codes, information theory and elliptic curve cryptography (ECC). Elliptic curve cryptography (ECC)can be used in various devices, such as wearable devices, key agreement and bank account systems. On one side, cryptographic system and algorithm should be high resistable to reduce the potential attacks, on the other side, the complexity of the cryptographic system shall be reduced. Basically, there are two bases, polynomial basis [5-13] and normal basis [14-17], which can be selected to represent the field operation. Nevertheless, in hardware realization, polynomial basis multipliers are more widely used compared to normal basis multipliers [8].

All-one-polynomials (AOP)s and trinomials are two of the important irreducible polynomials being used [7-11], [17-27]. Due to security reasons, AOPs are usually not preferred for cryptosystem implementations though the AOP-based multipliers are quite simple and regular, while trinomial-based multipliers are more popular than AOP-based ones, as two trinomials have been recommended by the National Institute of Standards and Technology (NIST) for ECC implementation [5]. However, because of the complexity

differences, AOPs and trinomials usually are not considered together in practical field multiplication implementations [18].

There are basically two kinds of structures for multipliers over $GF(2^m)$: systolic design and non-systolic design. Systolic multipliers over $GF(2^m)$ based on irreducible polynomials are preferred in high-performance applications due to their features such as modularity and regularity [5-11]. Systolic structures also have high register-complexity since all processing elements (PEs) in the systolic array need to use registers for pipelining [5], while non-systolic designs usually have lower complexity with larger critical-path delay.

For practical applications, especially in field-programmable gate array (FPGA) platforms, where the register-resources are not that abundant, low register-complexity systolic structures are required. Many efforts have been reported to reduce the register-complexity in systolic multipliers based on irreducible AOPs and trinomials [7-10], [23-27]. A bit-parallel AOP-based systolic multiplier has been introduced in [23]. Furthermore, another efficient AOP-based design is presented in [24]. Moreover, one low-complexity systolic Montgomery AOP-based multiplier has been proposed in [26]. In [7], Lee *et al.* presented a bit-parallel systolic trinomial multiplier. Meher [8] proposed efficient bit-parallel systolic and super-systolic designs. Xie *et al.* [9] introduced a low register-complexity systolic structure. Very recently, Montgomery systolic multipliers were presented where the register count was efficiently reduced [10]. Several other works were reported for efficient realization of finite field Montgomery multiplication over $GF(2^m)$ [11], [17].

In this thesis, we combine low register-complexity and Montgomery multiplication algorithm together to speed up the multiplication process.

1.2 Summery of contribution

1.2.1 Existing AOP Systolic Multiplier Based on Trinomial

There are some designs about finite field systolic multiplier based on trinomial have been reported. Most of these designs focus on the way to design the PEs inside of multiplier based on the critical-path. In this paper, we suggest two kinds of structure with critical paths of $T_A + T_X$ and $_{MAX}\{T_A, T_X\}$, where the duration of each cycle period is $T_A + T_X$ (T_A and T_X refer to the delay of an AND gate and a XOR gate, respectively). The critical-path of the second structure is shorter than the first one, so it has lower latency.

But the second multiplier needs more registers.

1.2.2 Proposed Systolic Structure

The efficiency of multiplier will be limited if we only use one algorithm. So the main contribution of this thesis needs to combine another novel Montgomery Algorithm together to improve the overall efficiency. Inside of the structure, we can apply the strategies of registers sharing and parallel-array pipelining to decompose the linear systolic design into several parallel arrays. According to the characteristics of one the inputs' matrix, we can observe that each two adjacent columns have mostly the same elements. Correspondingly, every two adjacent PEs can share the same input operands. In this way, we not only can decrease the latency and amount of registers but also can decrease the number of XOR2 gates. In order to confirm the proposed design, we choose $m = 233$ which has been recommended by NIST [15].

As we all know, the NAND gate is usually faster than the AND gate, so we use NAND2 to instead all AND2 in original structures. In order to satisfy the logic functions, we also need to change the XOR2 gate to XNOR2. Besides that, there is a need to redesign the wire connections between each shift unite. After changing these components, we can realize the same function with lower latency and lower register-complexity circuits. We use an example which $m = 162$ to test the proposal in this thesis.

1.2.3 Report Outline

The following parts of the report are organized in this way.

Chapter 2 shows the processes of mathematical formulation of polynomial basis multiplication over $GF(2^m)$. Several classes of irreducible polynomials are shown within this chapter.

Chapter 3 talks about the polynomial basis multiplication over $GF(2^m)$, and we show some examples about how to implement multiplication on finite field.

Chapter 4 presents a low register-complexity AOP-based systolic multipliers. There are some AOP-based multipliers, but they have high complexitites. In this chapter, we introduce two new AOP multipliers which have lower complexity of components.

Chapter 5 introduces how to use the proposed AOP multiplier as core component to realize trinomial multiplier. Besides this, in this chapter, we apply a new algorithm to

change the structure of previous multiplier which significantly improves the speed and area complexity performance. The structure is also improved by using digital-parallel way.

Chapters 6 and 7 present the conclusion for whole thesis and future plan.

Chapter 2

Finite Field

This chapter presents some basic mathematical background knowledge about finite field. In the following sections, we introduce the definition of field and finite field and algorithm operations in finite field. Besides this, we introduce a special finite field named Binary Field, as well as those field operations based on this field.

2.1 Introduction of Finite Field

2.1.1 Field

A field (f,+,·) is consisted by a set F with two operations which are addition (denoted by +) and multiplication (denoted by ·). And these two operations also satisfy the usual arithmetic properties, such as distributive law $(a + b) \cdot c = a \cdot c + b \cdot c$ for $a,\ b,\ c \in F$. Fields are abstractions of number systems and their essential properties, such as the real number R and the complex numbers C.

2.1.2 Finite Field

Finite field(also called Galois field) is a field which contains finite number of elements, and the order of a finite field is the number of elements it contains. Here is an example, if a field has q elements, then we can express it as F_q(also can be denoted as $GF(q)$,or Z/qZ),represented as the integers modulo q, the order of the field is q.

2.2 Field Operation

Addition and multiplication are two operations which can be involved in field F. So for subtraction and division, we need to use addition and multiplication to define them. We define subtraction in terms of addition: $a - b = a + (-b)$, for $a,\ b \in F$, and $-b$ is the unique element in F ($-b$ is the negative of b, $b + (-b) = 0$). Similar for division, division is defined in terms of multiplication: $a/b = a \cdot b^{-1}$, for $a,\ b \in F$, $b \neq 0$, and b^{-1} is a unique element in F (b^{-1} is the inverse of b, $b \cdot b^{-1} = 1$).
Here is an example of arithmetic operation in finite field.
The elements of $GF(26)$ are 0,1,2...,25, then the arithmetic operations in $GF(26)$ are:

(1). Addition:15+20=9 since 35 mode 26=9.

(2). Subtraction:15-20=15+(-20)=21 since -5 mode 26=21.

(3). Multiplication:$15 \cdot 20 = 14$ since 300 mode 26=14.

2.3 Binary Field

If the finite field with the the order 2^m is called binary field, the integer m is called the degree of the field. There are several ways to construct $GF(2^m)$. Such as the binary field $GF(2^m)$ can be consisted by 2^m possible bit strings with length m. For example

$$GF(2) = \{0, 1\}$$

$$GF(2^3) = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

The addition and multiplication are as follows:
Addition:

$$0 + 0 = 0,\ 0 + 1 = 1,\ 1 + 1 = 0$$

The addition of two elements should be optimized by bitwise addition modulo 2.
Such as (11000) + (10111) = (01111)
Multiplication:

$$0 * 0 = 0,\ 0 * 1 = 0,\ 1 * 1 = 1$$

Or we can use other two ways to construct $GF(2^m)$, one way is to use polynomial basis representation, the other way is to use normal basis representation. We are going to introduce polynomial basis representations in the next chapter.

GF(2)is the simplest finite field, it is consisted only by two elements, 0 and 1. For addition and multiplication in GF(2), we need to modulo 2. Depending on the results, the addition is equivalent to logical XOR, the multiplication is equivalent to logical AND.

2.4 Extension Field

Let A be a filed and B be subfield of field A. If C is subset of field A, then the field A is the extension field of field $B(C)$(which contain both B and C), which can be denoted as A/B. For instance, the rational numbers' extension field is real numbers, and the real numbers' extension field is complex numbers. Finite fields $GF(2^m)$ are the extension field of $GF(2)$.

Chapter 3

Polynomial basis multiplication over $GF(2^m)$

3.1 Polynomials

Assume there has field F, and the elements $a_n, a_{n-1}, a_{n-2}..., a_1, a_0$ belong to field F. Then the expression with the form of $f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2}... + a_2x^2 + a_1x + a_0$ is called a polynomial with degree n over F. The element a_i is called the coefficient of x^i in f(x), and $a_n \neq 0$. Depend on the each power of X and corresponding coefficient we can justify if this two polynomials are equal or not.

Polynomial ring R[X]is a ring which can be formed by the set of polynomials in one or more variables(such as x) with coefficients in another ring R(or field). Such as in X over a field P is defined as the set of expressions with the form $f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2}... + a_2x^2 + a_1x + a_0$, where $a_n, a_{n-1}, a_{n-2}..., a_1, a_0$ are the coefficients which are the elements of field P, then these coefficients can forma ring,called polynomial ring P[x]. Polynomials can be equipped with arithmetic operations. Two standard operations for polynomials are addition and multiplication. Here is an example. Let $A(x) = x^4+3x^2+2$ and $B(x) = 3x^2 + x + 4$ be elements of polynomial ring $R_4[x]$. The multiplications and addition of this two polynomials are:

$$A(x) \cdot B(x) = 3x^6 + x^5 + x^4 + 3x^3 + 2x^2 + 2x + 4$$

$$A(x) + B(x) = x^4 + 2x^2 + x + 2$$

3.2 Polynomial basis representation over $GF(2^m)$

The way to use polynomial basis representation to construct binary field $GF(2^m)$ means the elements of $GF(2^m)$ are binary polynomials with the degree at most $m-1$. In this condition the polynomials' coefficients are in the field of $GF(2)$. Such as, for finite field $GF(2^m)$, the elements in this field are the polynomials$\{0, 1, x, x+1, x^2, x^2+1, ..., x^{m-1}+x^{m-2}+...+x+1\}$, where the x is a root of an irreducible polynomial $f(\alpha)$ over $GF(2)$, and the polynomial coefficients are $GF(2) = \{0, 1\}$, where $f(x) = 0$.

We can use an example to show the exactly elements for finite field based on polynomials. The elements of finite field $GF(2^3)$ are as follows

Elements in $GF(2^m)$	Polynomial	Coordinates
0	0	(0,0,0)
x	1	(0,0,1)
x^2	x	(0,1,0)
x^3	$x+1$	(0,1,1)
x^4	x^2	(1,0,0)
x^5	x^2+1	(1,0,1)
x^6	x^2+x	(1,1,0)
x^7	x^2+x+1	(1,1,1)

3.3 Irreducible polynomial

3.3.1 Definition of irreducible polynomial

The irreducible polynomial means if a polynomial($f(x)$) couldn't be factored into the product of two non-constant polynomials over the same field($f(x) \neq g(x)h(x)$). Whether the polynomial can be factored or not should depend on the field and ring to which the coefficients are considered belong to. For instance, the polynomial $f(x) = x^2 - 2$ is irreducible if the coefficients -1 and 2 are considered belong to integers. But if we consider the coefficients are belong to real numbers, then the polynomial $f(x)$ can be factored as $(x+\sqrt{2})(x-\sqrt{2})$.

3.3.2 Important irreducible polynomial based on finite field

Irreducible polynomial also can be called as primitive polynomials. The irreducible polynomial can be used to represent the elements of a finite field. For example, there is an irreducible polynomial over GF(2) with degree m $f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2}... + f_1x + f_0$. Let $\alpha \in GF(2^m)$ be the root of this irreducible polynomial, then the set $\{1, \alpha, \alpha^2, ...\alpha^{m-2}, \alpha^{m-1}\}$ can constitute the polynomial basis in $GF(2^m)$. So these polynomial basis can be used to represented the elements in $GF(2^m)$ with the most degree as $m-1$, and the form is $GF(2^m) = \{a(x)|a(x) = a_{m-1}x^{m-1} + a_(m-2)x^{m-2} + ... + a_2x^2 + a_1x + a_0, a_j \in GF(2)\}$.

There are several different kinds of polynomial, and we usually choose all one polynomials(AOP), trinomials, and pentanomials to represent the elements of finite field. In this paper, our design is based on AOP and trinomial.

All One Polynomials(AOP)

All one polynomial (AOP) means the polynomial which all coefficients are one, and over the finite field $GF(2)$. AOP is irreducible polynomial only if $m+1$ is prime and 2 is a primitive modulo $m+1$. For example, the value of m should be $m \in \{2, 4, 6, 10, 12, ...\}$ The form of AOP can be written as

$$f(x) = \sum_{i=0}^{m} x^i = x^m + x^{m-1} + x^{m-2} + ... + x^2 + x + 1 \tag{3.1}$$

The degree is m ($m+1$ should be prime number). AOP has simple form, so it usually be used to define efficient algorithm and implement multiplication.

Trinomial

Trinomial is a polynomial consisting of three non-zero terms. We usually use trinomial in mathematics. Such as

- $5x + y + 6z$, where x,y,z are variables
- $m^2 + 4n + p$, where m,n,p are variables

- $Ax^m + Bx^n + Cx^p$, where x is variable, m,n,p are nonnegative integers

Trinomials over finite field are widely applied in different areas. Such as using trinomial to implement multiplier over finite field. And this kind of multiplier is the core component for ECC(Elliptic Curve Cryptography). The form of trinomial over$GF(2^m)$ is
$f(x) = x^m + x^k + 1$
The National Institute of Standard and Technology (NIST) [15] has recommend five binary finite fields for ECC implementation. There are two binary fields generate trinomials, $f(x) = x^{233} + x^{73} + 1$, and $f(x) = x^{409} + x^{87} + 1$.

3.4 Polynomial Basis Multiplication Over $GF(2^m)$

There is a irreducible polynomial over $GF(2)$
$f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + ... + f_2x^2 + f_1x + f_0$, $f_i \in GF(2)$=$\{0,1\}$
The set $\{1, x, x^2, ...x^{m-2}, x^{m-1}\}$ is the polynomial basis over finite field $GF(2^m)$. Comparing characters of this set and the form of polynomial, we can use polynomial to represent the elements of $GF(2^m)$; that is, we can represent the elements of this set by defining f(x) as
$\beta(x) = \beta_{m-1}x^{m-1} + \beta_{m-2}x^{m-2} + ... + \beta_2x^2 + \beta_1x + \beta_0$,where $\beta_i \in GF(2)$.
Let a(x) and b(x) be two field elements, and c(x)is the product of them, then we can have
$c(x) = a(x)b(x) \mod f(x)$
So, the polynomial basis multiplication is consisted by two steps: one is polynomial multiplication and another reduction modulo an irreducible polynomial. The product of d(x)=a(x)b(x) is a polynomial with degree $2m - 2$, so after the modular reduction $c(x) = a(x)b(x) \mod f(x)$, the degree $2m - 2$ of polynomial d(x)is reduced by degree m irreducible polynomial f(x). The multiplication process can be represented by matrix as follows(fig. 3.1, fig. 3.2),
So the process of $C = A \cdot B \ mod \ f(x)$ can be represented as follows(fig. 3.3)

3.4.1 AOP Basis Multiplication Over $GF(2^m)$

There is a irreducible AOP $f(\alpha)$ of degree m over GF(2)
$f(\alpha) = \alpha^m + \alpha^{m-1} + \alpha^{m-2} + ... + \alpha^2 + \alpha + 1$

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{m-2} \\ d_{m-1} \\ d_m \\ d_{m+1} \\ \vdots \\ d_{2m-3} \\ d_{2m-2} \end{pmatrix} = \begin{pmatrix} a_0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_2 & a_1 & a_0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-2} & a_{m-3} & a_{m-4} & a_{m-5} & \cdots & a_0 & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & a_{m-4} & \cdots & a_1 & a_0 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & a_{m-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{pmatrix}$$

Figure 3.1: d(x)=a(x)b(x)

$$\begin{pmatrix} c_0 \\ c_1 \\ \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & r_{0,0} & r_{0,m-2} \\ 0 & 1 & 0 & r_{1,0} & r_{1,m-2} \\ & & & & \\ 0 & 0 & 1 & r_{m-1,0} & r_{m-1,m-2} \end{pmatrix} \begin{pmatrix} d_0 \\ \\ d_{m-1} \\ d_m \\ \\ d_{2m-2} \end{pmatrix}$$

Figure 3.2: c(x)=d(x)mod f(x)

$$[A^0 \quad A^1 \quad A^2 \quad . \quad . \quad . \quad A^{m-2} \quad A^{m-1}] * \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ . \\ . \\ . \\ b_{m-2} \\ b_{m-1} \end{bmatrix} = \begin{bmatrix} C^0 \\ C^1 \\ C^2 \\ . \\ . \\ . \\ C^{m-2} \\ C^{m-1} \end{bmatrix}$$

Figure 3.3: $C = A \cdot B \ mod \ f(x)$

where $m+1$ is prime number,and 2 is the primitive modulo of $m+1$. Assume x is a root of $f(\alpha)=0$, then we can have

$f(x)+xf(x)=0$,

that means, $(x^m+x^{m-1}+x^{m-2}+...+x^2+x+1)+x(x^m+x^{m-1}+x^{m-2}+...+x^2+x+1)$

$=x^m+x^{m-1}+x^{m-2}...+x^2+x+1+x^{m+1}+x^m+x^{m-1}+...+x^3+x^2+x$ (1)

because in finite field GF(2), $1+1=0, 1+0=1, 0+0=0$ thus for function(1),

$(1)=x^{m+1}+1=0$

and we can have

$x^{m+1}=1$

We define $\{x^{m+1}, x^m, x^{m-1}, ..., x^2, x, 1\}$ as the extended polynomial basis. For any elements A,B,C belong to finite field $GF(2^m)$, they can be represented in above extended polynomial basic

$$A=\sum_{j=0}^{m} a_j x^j, \tag{3.2}$$

$$B=\sum_{j=0}^{m} b^j x^j, \tag{3.3}$$

$$C=\sum_{j=0}^{m} c^j x^j, \tag{3.4}$$

where $a_j, b_j, c_j \in GF(2)$.

We define C is the product of A multiply B, then we can have

$C=A\cdot B \mod f(x)$

because

$$B=\sum_{j=0}^{m} b^j x^j \tag{3.5}$$

so we can have

$$C=\sum_{j=0}^{m} b_j(x^j\cdot A \mod f(x)) \tag{3.6}$$

if we define $A^0=A$ and $A^i=x^i\cdot A \mod f(x)$, the form of A^i is

$$A^i=\sum_{j=0}^{m} a_j^i x^j \tag{3.7}$$

then we can represent A^{i+1} with A^i as

$A^{i+1} = x \cdot A^i \mod f(x) = (a_0^i x + a_1^i x^2 + ... + a_m^i \cdot x^{m+1}) \mod f(x)$

where $a_0^{i+1} = a_m^i, a_j^{i+1} = a_{j-1}^i (1 \leq j \leq m-1)$

Depend on above equations we also can get

$$a_j^{i+l} = \begin{cases} a_{m-l+j+1}^i & \text{if } 0 \leq j \leq l-1 \\ a_j^i - l & \text{otherwise} \end{cases}$$

3.4.2 Trinomial Basis Multiplication Over $GF(2^m)$

Let we define a finite filed $GF(2^m)$, usually the form of irreducible polynomial of degree m is

$$f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + ... + f_2x^2 + f_1x + f_0 \tag{3.8}$$

where $f_i \in GF(2) = \{0, 1\}$ and $\{f_i, for 1 \leq i \leq m-1\}$

Assume α is a root of polynomial $f(x)$. The polynomial basis $\{1, \alpha, \alpha^2, ..., \alpha^{m-2}, \alpha^{m-1}\}$ can be represented by irreducible polynomial. Let A,B be two elements in finite field $GF(2^m)$, so A, B can be represented as follows

$$A = \sum_{j=0}^{m-1} a_j \alpha^j, \tag{3.9}$$

$$B = \sum_{j=0}^{m-1} b^j \alpha^j, \tag{3.10}$$

for which $a_j, b_j \in GF(2)$, and $0 \leq j \leq m-1$

Let C be the product of A and B

$$C = A \cdot B \mod f(x)$$

$$B = \sum_{j=0}^{m-1} b^j \alpha^j, \tag{3.11}$$

so in order to derive recurrence relation for multiplication, C can be represented as

$$C = \sum_{i=0}^{m-1} b_i(\alpha^i \cdot A \mod f(x)) \tag{3.12}$$

We can let $Q_i = b_i A^i$, $A^0 = A$ and $A^i = \alpha^i \cdot A \mod f(x)$then the equation (3.11) can be expressed as another form

$$C = \sum_{i=0}^{m-1} Q_i \tag{3.13}$$

$$C = Q_0 + Q_1 + Q_2 +, , , + Q_{m-3} + Q_{m-2} + Q_{m-1}, \tag{3.14}$$

$$Q_i = b_i A^i, Q_{i+1} = b_{i+1} A^{i+1} \tag{3.15}$$

A^{i+1} can be obtain from A^i recursively as:

$$A^{i+1} = \alpha \cdot A^i \mod f(x) \tag{3.16}$$

The equation (3.12) reflect the addition of reduced polynomial. The equation (3.14) reflect the partial product generation. The equation (3.15) reflect the modular reduction. For the process of adding, we can use XOR gate for our product.
From the above derivation, we get equation $A^i = \alpha^i \cdot A \mod f(x)$

$$A^i = [a_0^i \alpha^0 + a_1^i \alpha^1 + a_2^i \alpha^2 + ... + a_{m-2}^i \alpha^{m-2} + a_{m-1}^i \alpha^{m-1}] \mod f(x) \tag{3.17}$$

Therefor, for polynomial equation (3.15), the right side can ne expanded as

$$A^{i+1} = \alpha \cdot A^i \mod f(x) \tag{3.18}$$

$$= [\alpha(a_0^i + a_1^i \alpha + a_2^i \alpha^2 + ... + a_{m-2}^i \alpha^{m-2} + a_{m-1}^i \alpha^{m-1})] \mod f(x) \tag{3.19}$$

$$= [a_0^i \alpha + a_1^i \alpha^2 + a_2^i \alpha^3 + ... + a_{m-2}^i \alpha^{m-1} + a_{m-1}^i \alpha^m] \mod f(x) \tag{3.20}$$

We mentioned before α is a root of polynomial of $f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + ... + f_2 x^2 + f_1 x + f_0$, where $f_i \in GF(2) = \{0, 1\}$ and $\{f_i for 1 \leq i \leq m-1\}$.
Thus,

$$f(\alpha) = \alpha^m + f_{m-1}\alpha^{m-1} + f_{m-2}\alpha^{m-2} + ... + f_2\alpha^2 + f_1\alpha + f_0 = 0 \tag{3.21}$$

$$\alpha^m = f_{m-1}\alpha^{m-1} + f_{m-2}\alpha^{m-2} + ... + f_2\alpha^2 + f_1\alpha + f_0 \tag{3.22}$$

Substituting equation (3.21) into equation (3.19), then the equation of (3.19) can be obtained as

$$A^{i+1} = [a_0^{i+1} + a_1 i + 1\alpha + ... = a_{m-2}^{i+1}\alpha^{m-2} + a_{m-1}^{i+1}\alpha m - 1] \mod f(x) \tag{3.23}$$

Compare the equation(3.22) with the equation (3.16), we can observe the transitions from A^i to A^{i+1} are

$$a_0^{i+1} = a_{m-1}^i \tag{3.24}$$

$$a_j^{i+1} = a_{j-1}^i \oplus f_j \cdot a_{m-1}^i, \tag{3.25}$$

where $j = 1, 2, 3, ..., m-1$

In this paper, the multiplier over finite field $GF(2^m)$ that we design is based on trinomial. Assume there is an irreducible trinomial with degree m, the form is

$$f(x) = x^m + x^n + 1 \tag{3.26}$$

Comparing equation (3.26) and (3.8), we can find

$$f_j = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } 1 \le j \le m-1 \text{ and } j \ne k \end{cases}$$

By using these values of f_i, the equations of modular reduction from (3.23) to (3.25) can be written as

$$a_0^{i+1} = a_{m-1}^i, \tag{3.27}$$

$$a_k^{i+1} = a_{k-1}^i \oplus a_{m-1}^i, \tag{3.28}$$

$$a_j^{i+1} = a_{j-1}^i, for 1 \le j \le m-1 and j \ne k \tag{3.29}$$

The logic equations from (3.27) to (3.29) can reflect the recursive process of modular reduction from A^i to A^{i+1}. Further more, we can extend these equations to have higher order modular reduction. Assume we need to operand from A^i to A_{i+l}, which $1 \le l \le m-k$. Then the logic equations can be written as

$$a_j^{i+l} = \begin{cases} a_{m-l+j}^i & \text{for } 0 \le j \le l-1 \\ a_{j-l}^i & \text{for } k \le j \le k+l-1 \\ a_{j-l}^i & \text{otherwise} \end{cases}$$

3.5 Existing Researches About Polynomial Basis Multiplication Based On Finite Field $GF(2^m)$

For finite field algorithm, the most important application is in cryptograph, such as Elliptic Curve Cryptography (ECC) and pairing based cryptography. The multiplier based on polynomial over finite field is the main component for ECC.

The multiplier is easy to be implemented on a software platform. But for the real practical applications, such as credit card and cellphone, it's not easy to embed software platform in these devices. Except this, the software platform is also hard meet the requirement of speed and time critical systems. In order to satisfy the specific requirements, we need the hardware implementations in cryptographic systems.

The hardware implementation usually has two key points need to be paid attention, area and timing. We prefer the devices that we use can be faster and lighter, such as cellphone. The iphone is usually popular than other cellphone, one of the important reason is iphone has faster speed than most cellphones. If a device should be built by a lot of components, then this device will have larger area and higher cost. If we want to get more profits from what we design, we need to save the sources. Therefore, high speed and smaller area are already become two important demands of design.

As the development of technologies and algorithms, there are some efficient multipliers based on finite field have been realized. In structure part, we apply four main structures: (1)parallel-in serial-out(PISO), (2)serial-in parallel-out(SIPO),(3)parallel in parallel out, (4)serial/prallel structures. The parallel-in serial-out structures has smaller area but has low throughput. The serial-in parallel-out structure has higher throughput, but it need larger area. For hardware implementation, we usually has critical requirement for timing and area. There are some works can achieve this optimal balance. We can divide these works in to two types: systolic structure and non-systolic structure. The systolic structure can accomplish the critical timing requirement and has the features of regularity and modularity. Except this, for systolic design, all processing elements (PEs) are fully pipelined to produced very high throughput rate. For non-systolic design, it usually has low latency but it has has low throughput. The relative pictures will be shown below.

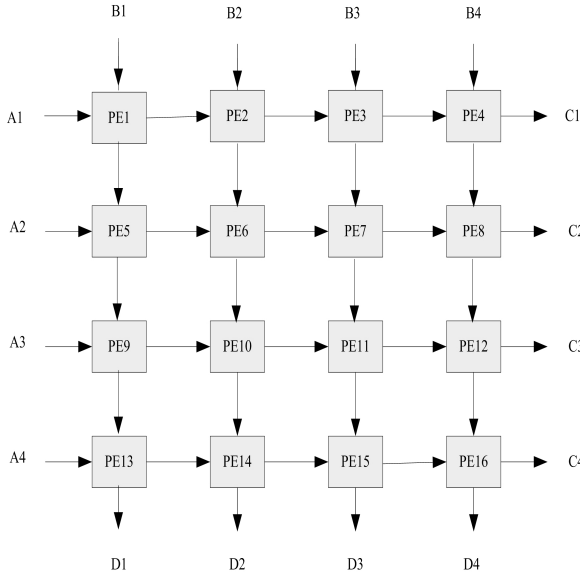

Figure 3.4: systolic array

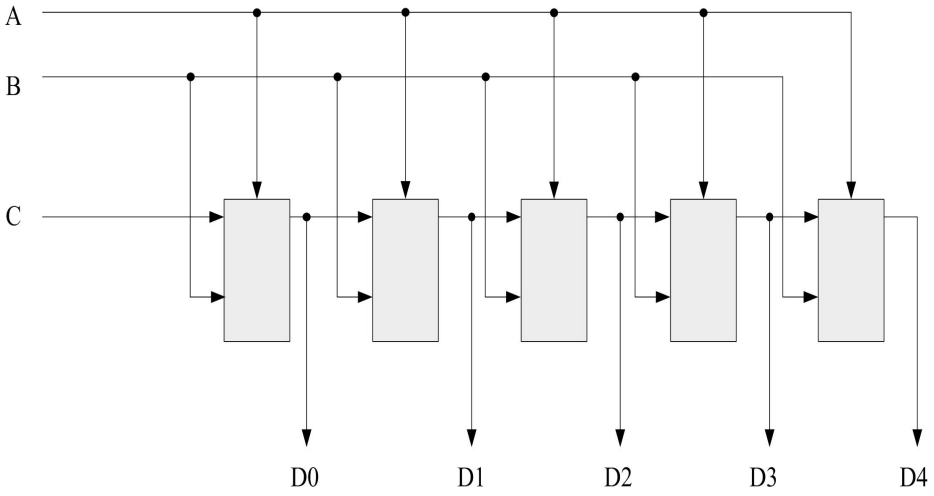

Figure 3.5: serial in parallel out structure

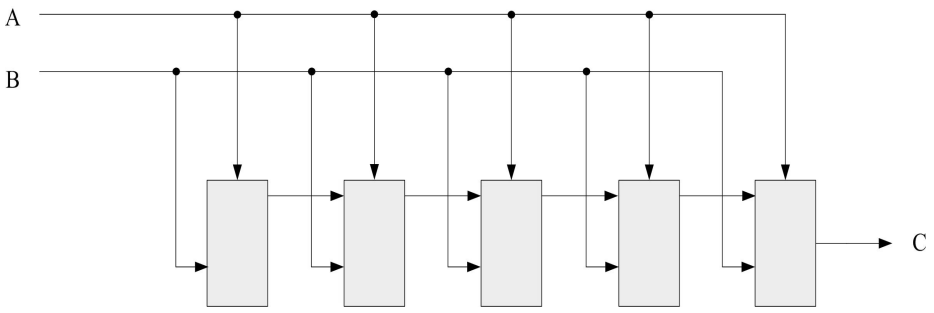

Figure 3.6: parallel in serial out structure

3.5.1 Existing Example For AOP

In paper [7], the author introduced the systolic and super-systolic multiplier for $GF(2^m)$ based on trinomial. In this paper, the author analysed what is systolic structure, and how to implement multiplication based on trinomial. Combine with appropriate cut-set retiming and bit level pipeline systolic structure, the latency can be decreased.

3.5.2 Existing Example For Trinomial

AOP has simpler form so we like to use it to implement some simple structures, but we don't prefer multiplier based on AOP as one of the component for ECC. In paper [16], the authors present an efficient recursive formulation and use systolic structure to implement the multiplier based on AOP. This new multiplier is finished by efficient way that can decrease the number of registers and latency.

Chapter 4

Low Register-Complexity AOP based Systolic Multipliers (AOP-Based Computation Core)

In this section, we briefly review the AOP-based multiplication algorithm first, and then present our proposed architectures based on the existing structures.

4.1 Review of AOP Multiplication Algorithm [24]

For simplicity of discussion, let $f(\alpha) = \alpha^k + \alpha^{k-1} + \cdots + \alpha + 1$, be an irreducible AOP of degree k over $GF(2)$ (where $k+1$ is prime and 2 is the primitive modulo $k+1$). For any $x \in GF(2)$, and x is a root of $f(\alpha) = 0$, we have

$$\begin{aligned} f(x) + xf(x) &= (x^k + x^{k-1} + \cdots + x + 1) \\ &+x(x^k + x^{k-1} + \cdots + x + 1) = x^{k+1} + 1 = 0, \end{aligned} \tag{4.1}$$

and then we have

$$x^{k+1} = 1. \tag{4.2}$$

Then, let $\{x^{k+1}, x^k, \ldots, x, 1\}$ be the extended polynomial basis [28]. For any $A, B, C \in GF(2^m)$, they can be represented in the extended polynomial basis as

$$A = \sum_{j=0}^{k} a_j x^j \tag{4.3}$$

$$B = \sum_{j=0}^{k} b_j x^j \tag{4.4}$$

$$C = \sum_{j=0}^{k} c_j x^j, \tag{4.5}$$

where $a_j, b_j, c_j \in GF(2)$, for $0 \leq j \leq k-1$, and $a_k = 0$, $b_k = 0$, and $c_k = 0$.

Let us define C as the product of A and B, and then we have

$$C = A \cdot B \bmod f(x), \tag{4.6}$$

which can be written into another form

$$C = \sum_{j=0}^{k} b_j (x^j \cdot A \bmod f(x)). \tag{4.7}$$

Let us define $A^0 = A$, and $A^i = x^i \cdot A \bmod f(x)$, such that A^{i+1} can be obtained from A^i as

$$A^{i+1} = x \cdot A^i \bmod f(x). \tag{4.8}$$

Then, we have

$$A^{i+1} = (a_0^i x + a_1^i x^2 + \cdots + a_k^i \cdot x^{k+1}) \bmod f(x), \tag{4.9}$$

where

$$A^i = \sum_{j=0}^{k} a_j^i x^j. \tag{4.10}$$

Then, we have

$$A^{i+1} = a_0^{i+1} + a_1^{i+1} x + \cdots + a_k^{i+1} x^k, \tag{4.11}$$

where

$$\begin{aligned} a_0^{i+1} &= a_k^i, \\ a_j^{i+1} &= a_{j-1}^i, \text{for } 1 \leq j \leq k-1. \end{aligned} \tag{4.12}$$

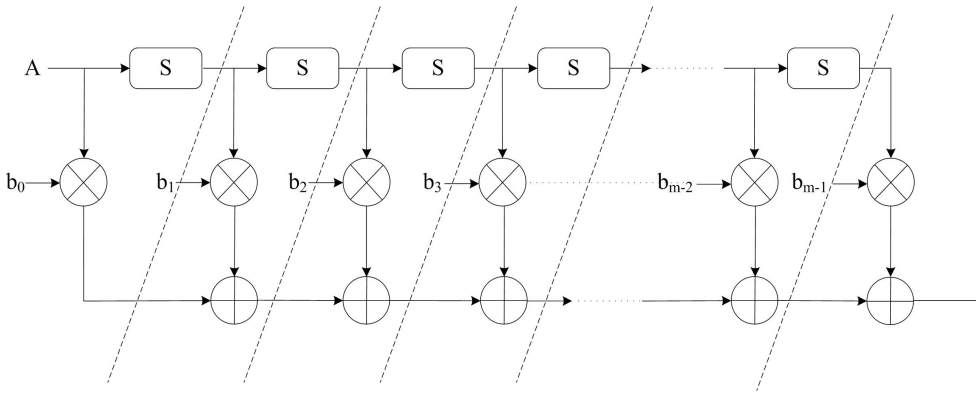

Figure 4.1: cut-set retiming of the SFG with $T_A + T_X$ critical path

One can also extend to obtain A^{i+l} from A^i for $1 \leq l \leq k$, such that

$$\begin{aligned} a_j^{i+l} &= a_{k-l+j+1}^i \qquad \text{for } 0 \leq j \leq l-1, \\ a_j^{i+l} &= a_{j-l}^i \qquad \text{otherwise.} \end{aligned} \tag{4.13}$$

4.2 Existing Systolic Structures

The conventional systolic structure based on the algorithm in Section II-A can be seen in Fig. 4.2 (structure-I: S-I), where it consists of $(k+1)$ PEs (including three types of PEs: PE-1, PE-2 and regular PE). The internal structures of these PEs are shown in Fig. 4.2. 1(b), (c), and (d), respectively, where BSC denotes the bit-shifting cell. The latency of the structure in Fig. 4.2 is $(k+1)$ cycles, where the duration of each cycle period is $T_A + T_X$ (T_A and T_X refer to the delay of an AND gate and a XOR gate, respectively). The way we do cut-set to get the critical path as $T_A + T_X$ is shown in Fig.4.1. A recent work has presented a low critical-path delay systolic structure (only T_X) [24], and it is shown in Fig. 4.4 (structure-II: S-II). The entire structure contains $(k+2)$ PEs, where the internal structures of PEs are shown in Figs. 4.4(b), (c), (d) and (e), respectively. The latency of this structure is $(k+2)$ cycles (critical-path delay: T_X). The way we do cut-set to get the critical path as $max\{T_A, T_X\}$ is shown in Fig. 4.1, usually $T_X > T_A$, so the critical path is T_X.

4.3 Modified Low Register-Complexity Structures

For the structures of Figs. 4.2 and 4.4, we find that k^2 registers in the PEs pipeline identical data (in shifted order) to the neighboring PEs. These registers can be removed

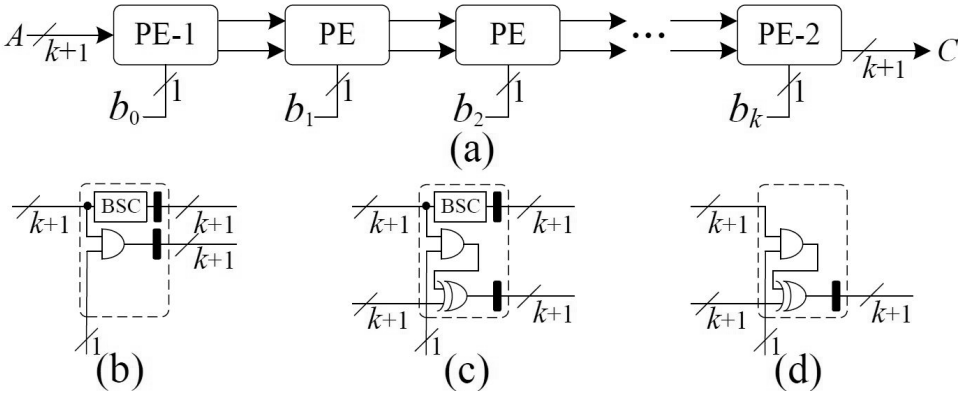

Figure 4.2: Conventional systolic structure of AOP-based multiplication (structure-I: S-I), where BSC denotes the bit-shifting cell and the black box denotes the registers. (a) Structure. (b) Internal structure of PE-1. (c) Internal structure of regular PE. (d) Internal structure of PE-2.

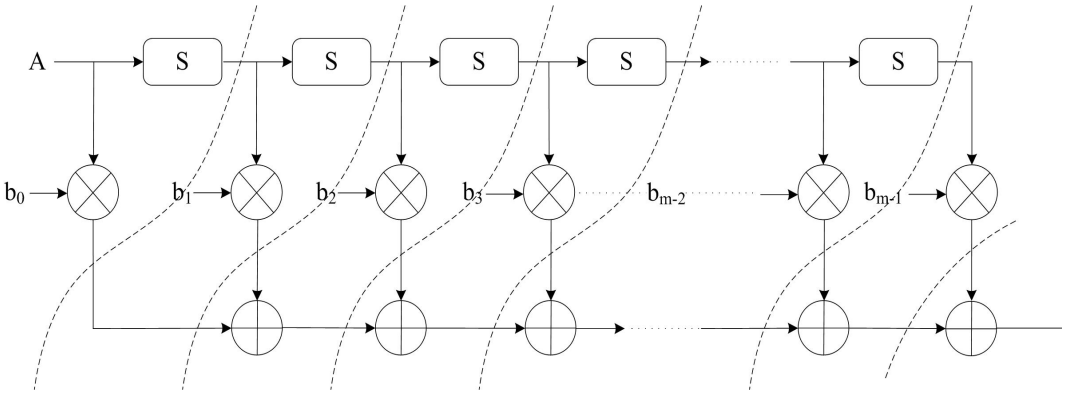

Figure 4.3: cut-set retiming of the SFG with $max\{T_A, T_X\}$ critical path

if we change the broadcasting strategy. As shown in Figs. 4.5 and 4.6, i.e., MS-I and MS-II, a shifted connection strategy is used that the input A is directly fed to each PE, and thus reduces the registers required previously. Moreover, the details of shifted connection is also shown in Figs. 4.5 and 4.6. To reduce the complexity further, we have used NAND and XNOR gates to replace the original AND and XOR gates,and also the speed of NAND and XNOR is faster than AND and XOR, as depicted in [7] and [11] (the critical-path is then shortened to $T_{NA} + T_{XN}$, where T_{NA} and T_{XN} represent the delay of NAND and XNOR gates, respectively). It is noted that for AOP-based multiplication, the last PE (inside the dotted box) can be removed as $b_k = 0$. The modified structures involve nearly the same time-complexity as the previous ones, but the register-complexity is significantly reduced.

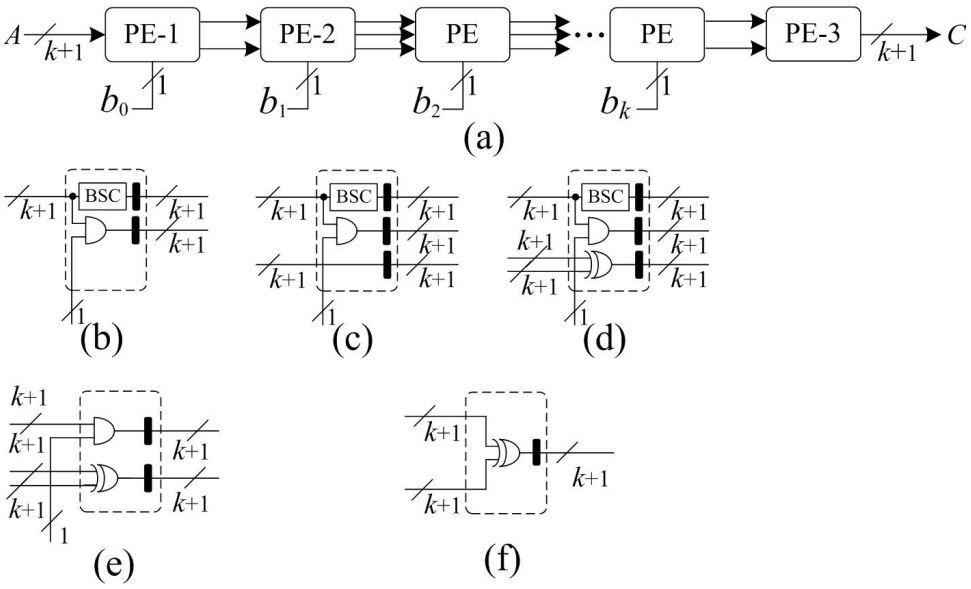

Figure 4.4: Existing low critical-path structure of [24] for AOP-based multiplication (structure-II: S-II), where the black box denotes the registers. (a) Structure. (b) Internal structure of PE-1. (c) Internal structure of PE-2. (d) Internal structure of regular PE. (e) Internal structure of PE-3. (f) Internal structure of PE-4

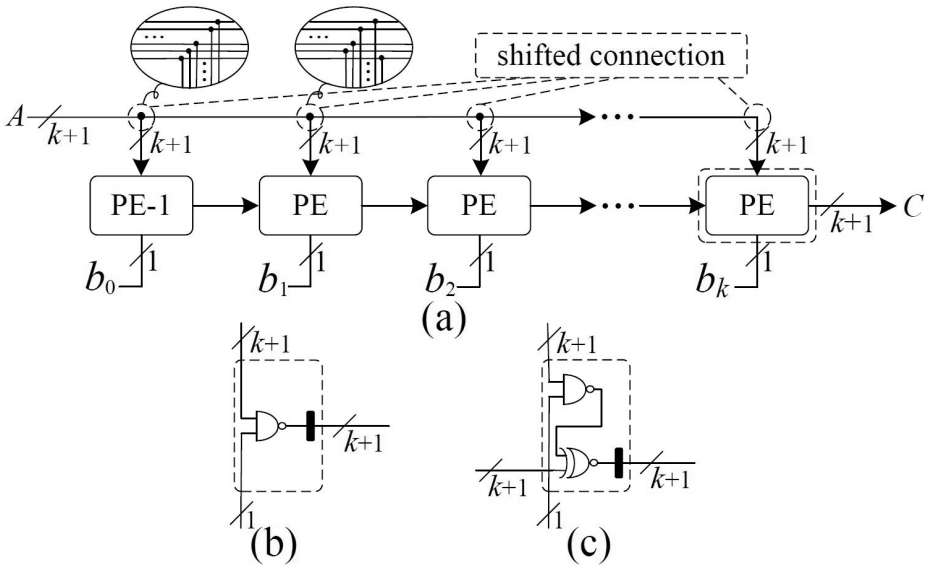

Figure 4.5: Modified structure-I (MS-I), where the black box denotes the registers. For AOP implementation, we can remove the PE inside the red-dotted area since $b_k = 0$, but for the formation of standard computation core, this PE will be preserved. (a) MS-I. (b) Internal structure of PE-1. (c) Internal structure of regular PE.

4.4 Low Latency Implementations

For practical applications, we can further reduce the latencies of structures shown in Figs. 4.5 and 4.6, for $k + 1 = pq + f$, where $0 \leq f \leq q$. Without loss of generality, we assume

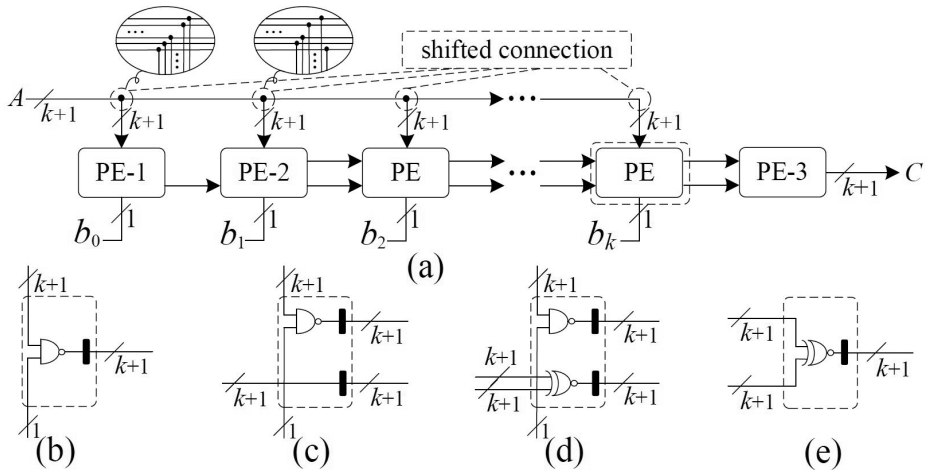

Figure 4.6: Modified structure-II (MS-II), where the black box denotes the registers. For AOP implementation, we can remove the PE inside the red-dotted area since $b_k = 0$, but for the formation of standard computation core, this PE will be preserved. (a) MS-II. (b) Internal structure of PE-1. (c) Internal structure of PE-2. (d) Internal structure of regular PE. (e) Internal structure of PE-3.

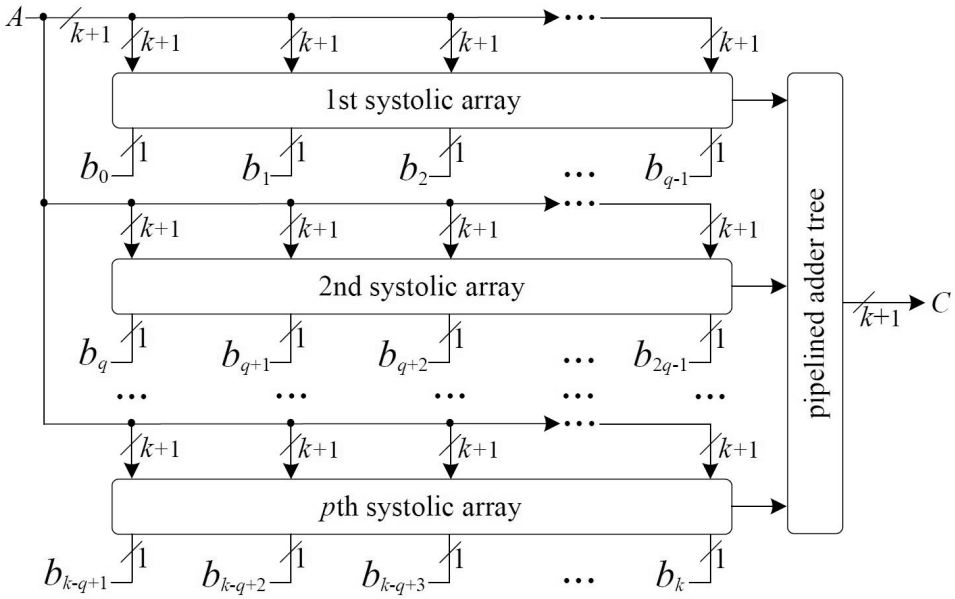

Figure 4.7: Low latency implementation of systolic structure, where the internal PEs can be those of MS-I or MS-II.

$f = 0$, and then we can decompose the original one systolic array of $k + 1$ PEs into p parallel arrays to achieve low latency implementations, as shown in Fig. 4.7. An extra pipelined adder tree consisting of XNOR gates(m is even) and registers is needed to add the results from p arrays together to yield final result C.

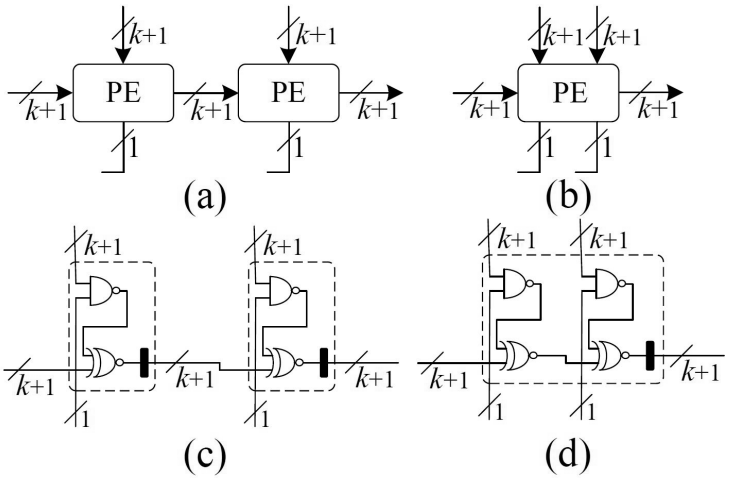

Figure 4.8: PE design for digit-parallel implementation ($d = 2$, based on the PEs from MS-I). (a) Original two neighboring PEs. (b) Combined PE. (c) Internal structure of previous two PEs. (d) Internal structure of combined PE.

Table 4.1: COMPARISON OF AREA-TIME COMPLEXITIES OF VARIOUS SYSTOLIC AOP-BASED MULTIPLIERS

Design	AND	NAND	XOR	XNOR	Register	Critical-path delay	Latency
Fig. 4.2 (S-I)	$(k+1)^2$	0	k^2+k	0	$2k^2+3k+1$	$T_A + T_X$	$k+1$
Fig. 4.4 (S-II)	$(k+1)^2$	0	k^2+k	0	$3k^2+6k+3$	T_X	$k+2$
Fig. 4.5 (MS-I)	0	k^2+k	0	k^2-1	k^2+2k	$T_{NA} + T_{XN}$	k
Fig. 4.6 (MS-II)	0	k^2+k	0	k^2-1	$2k^2+2k$	T_{XN}	$k+1$
Fig. 4.7 [1]	0	k^2+k	0	$\simeq k^2-1$	$\simeq k^2+2k$	$T_{NA} + T_{XN}$	$h+\log_2 p$
Digit-parallel[2] ($d=2$)	0	k^2+k	0	$\simeq k^2-1$	$\simeq k^2/2+k$	$T_{NA} + T_{XN}$	$(h+\log_2 p)/2$

[1]: Based on structure of Fig. 4.5 (MS-I).
[2]: Based on low-latency structure of Fig. 4.7 (MS-I).

4.5 Digit-Parallel Structures

We can combine neighboring PEs in a systolic array into one PE to reduce the register usage further. Fig. 4.8 shows an example of combining two neighboring PEs into one PE (based on the PEs from MS-I). The critical-path delay of the new PE thus turns into $(T_{NA} + 2T_{XN})$. For simplicity, we define the structure based on new PE in Fig. 4.8(b) as a digit-level parallel structure with digit-size $d = 2$. If we choose the value of d appropriately, the proposed architecture can achieve the optimal area-time complexity for specific application environments.

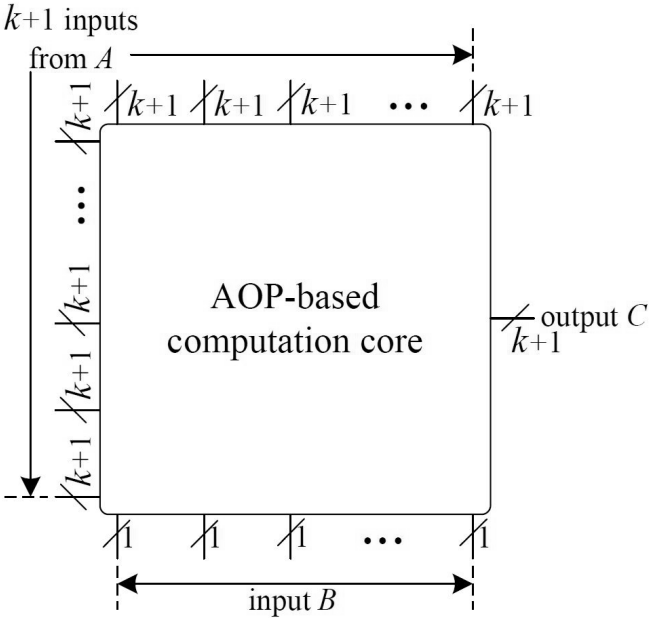

Figure 4.9: AOP-based standard computation core, where the internal PEs can be those of MS-I or MS-II (the internal structure can be as that of Fig. 5, based on specific application environment).

4.6 Area-Time Complexity

The area-time complexity of proposed designs in Figs. 4.5, 4.6, 4.7, and 4.8 are shown in Table 4.1 along with existing and conventional designs of Figs. 4.2 and 4.4. It can be seen that the proposed designs involve significantly less area-time complexity when compared with competing ones, especially on the register-complexity.

4.7 FPGA Implementation of Various AOP-based Structures

We have also implemented these AOP-based systolic structures to confirm the efficacy of proposed structures. We have synthesized these designs using Xilinx ISE 14.1 on Virtex 6 family device with $k = 162$. The results in terms of area-time-power complexity are shown in Table II. It can be seen that the proposed structures outperform the existing ones, especially for area-complexity. Since there is only minor difference between critical-paths of $T_{NA} + T_{XN}$ and T_{XN} in FPGA platforms, The proposed MS-II does not have significant advantage over existing ones. Therefore, the proposed MS-I can be used more widely than MS-II in practical applications.

Table 4.2: FPGA IMPLEMENTATION RESULTS OF VARIOUS AOP-BASED MULTIPLIERS FOR k=162

Design	Area	Delay[1]	Power	ADP[2]	PDP[3]
Fig. 4.2 (S-I)	53,300	154.035	2.236	8,210,066	344.42
Fig. 4.4 (S-II)	79,868	154.98	2.407	12,377,943	373.04
Fig. 4.5 (MS-I)	26,569	141.102	2.159	3,748,939	304.64
Fig. 4.6 (MS-II)	53,138	154.035	2.23	8,185,112	343.50

Unit for area: number of slice register; Unit for delay: ns; Unit for power: W (Power is estimated at 100MHz).
[1]: Delay = Latency.
[2]: ADP: Area-delay product = Area×Delay.
[3]: PDP: Power-delay product = Power×Delay.

4.8 AOP-based Computation Core

To fully utilize the special property of proposed AOP-based multipliers, we pack the structure of Fig. 4.7 (or combine with the structure of Fig. 4.8) as a standard computation core. The standard computation core is shown in Fig. 4.9, which consists of $k+1$ input bits from A, $k+1$ bits from input B, and $k+1$ bits of output C. For practical applications of this standard computation core, we can replace $k+1$ with any other integer. It is noted that both PEs of MS-I and MS-II can be used as internal structures for this computation core.

Chapter 5

Application of the Proposed AOP-based Computation Core

In this chapter, we focus on the application of the AOP-based computation core to obtain a low register-complexity Montgomery multiplication based on trinomials.

5.1 Montgomery Multiplication Algorithm

Montgomery multiplication is a method to perform fast modular multiplication, it was introduced by American mathematician Peter L. Montgomery in 1985. Montgomery multiplication is a way to transform the classical modular multiplication $a \cdot b \bmod N$ to Montgomery form $a \cdot b \cdot R \bmod N$).

Let $f(x)$ be a degree m irreducible trinomial over $GF(2)$ as

$$f(x) = x^m + x^n + 1. \tag{5.1}$$

where $1 \leq n \leq m-1$, such that we can have the Montgomery multiplication as [12]

$$C = A \cdot B \; r^{-1} \bmod f(x), \tag{5.2}$$

where A and B and their product C are elements in $GF(2^m)$ as

$$A = \sum_{i=0}^{m-1} a_i x^i, \tag{5.3}$$

$$B = \sum_{i=0}^{m-1} b_i x^i, \tag{5.4}$$

$$C = \sum_{i=0}^{m-1} c_i x^i, \tag{5.5}$$

for $\{a_j, b_j, \text{ and } c_j\} \in GF(2)$.

For r is the Montgomery factor that satisfies $gcd(r, f(x)) = 1$ (gcd refers to the greatest common divisor). Different algorithms have different selections of r to have the corresponding structures, as shown in [10-12]. In this algorithm, we have chosen $r = x^t = x^{(m-1)/2}$ (for NIST recommended trinomials, m is an odd number). Then, (5.2) can be expressed as

$$\begin{aligned} C &= A \cdot B \; r^{-1} \text{mod } f(x) \\ &= \sum_{i=0}^{m-1} b_i (A \cdot x^i \cdot x^{-t}) \text{mod } f(x) = C_1 + C_2, \end{aligned} \tag{5.6}$$

where

$$\begin{aligned} C_1 &= \sum_{i=0}^{t-1} b_i \cdot A \cdot x^{i-t} \text{mod } f(x), \\ C_2 &= \sum_{i=t}^{m-1} b_i \cdot A \cdot x^{i-t} \text{mod } f(x). \end{aligned} \tag{5.7}$$

For C_1, we define $A_1^{(0)} = A$, $A_1^{(1)} = A \cdot x^{-1} \text{mod } f(x)$, ..., $A_1^{(t)} = A \cdot x^{-t} \text{mod } f(x)$. Then, we have

$$A_1^{(i+1)} = A_1^{(i)} \cdot x^{-1} \text{mod } f(x) \tag{5.8}$$

where $0 \le i \le t-1$. C_1 can be expressed as

$$C_1 = \sum_{i=1}^{t} A_1^{(i)} b_{t-i}, \tag{5.9}$$

Define again

$$A_1^{(i)} = a_{1,0}^{(i)} + a_{1,1}^{(i)} x + \cdots + a_{1,m-1}^{(i)} x^{m-1}. \tag{5.10}$$

Then, we have

$$\begin{aligned} A_1^{(i+1)} &= a_{1,0}^{(i)} x^{-1} + a_{1,1}^{(i)} + \cdots + a_{1,m-1}^{(i)} x^{m-2} \\ &= a_{1,0}^{(i+1)} + a_{1,1}^{(i+1)} x + \cdots + a_{1,m-1}^{(i+1)} x^{m-1}. \end{aligned} \tag{5.11}$$

Since x is the root of $f(x) = x^m + x^n + 1$, we can have $x^m + x^n = 1$ and $x^{m-1} + x^{n-1} = x^{-1}$. Substituting them into (5.11) yields

$$
\begin{aligned}
a_{1,m-1}^{(i+1)} &= a_{1,0}^{(i)}, \\
a_{1,n-1}^{(i+1)} &= a_{1,n}^{(i)} \oplus a_{1,0}^{(i)}, \\
a_{1,j}^{(i+1)} &= a_{1,j+1}^{i},
\end{aligned}
\tag{5.12}
$$

for $0 \leq j \leq m-2$ and $j \neq n$. Similarly, for C_2, we can define $A_2^{(0)} = A$, $A_2^{(1)} = A \cdot x \mod f(x)$, ..., $A_2^{(t)} = A \cdot x^t \mod f(x)$ and $A_2^{(i+1)} = A_2^{(i)} \cdot x \mod f(x)$ (where $0 \leq i \leq t-1$). With these definitions, C_2 can be expressed as

$$
C_2 = \sum_{i=0}^{t} A_2^{(i)} b_{i+t}. \tag{5.13}
$$

Let us define again $A_2^{(i)} = a_{2,0}^{(i)} + a_{2,1}^{(i)} x + \cdots + a_{2,m-1}^{(i)} x^{m-1}$. Similarly, we have

$$
\begin{aligned}
a_{2,0}^{(i+1)} &= a_{2,m-1}^{(i)}, \\
a_{2,n}^{(i+1)} &= a_{2,n-1}^{(i)} \oplus a_{2,m-1}^{(i)}, \\
a_{2,j}^{(i+1)} &= a_{2,j-1}^{(i)},
\end{aligned}
\tag{5.14}
$$

for $1 \leq j \leq m-1$ and $j \neq n$.

5.2 Proposed Montgomery Multiplication Algorithm

The equations (5.1)-(5.14) represent the standard Montgomery multiplication process. To facilitate the Montgomery multiplication suitable for employing the proposed AOP-based computation core, we present the following proposed algorithm: Let x^m be an extended polynomial basis. From (5.11), we define

$$
A_U^{(1)} = \sum_{i=0}^{m} a_{U,i}^{(1)} x^i = a_{U,0}^{(1)} + a_{U,1}^{(1)} x + \cdots + a_{U,m}^{(1)} x^m, \tag{5.15}
$$

where

$$
\begin{aligned}
a_{U,0}^{(1)} + a_{U,1}^{(1)} x + \cdots + a_{U,m-1}^{(1)} x^{m-1} &= \sum_{i=0}^{m-1} a_i^{(0)} x^i, \\
a_{U,m}^{(1)} = a_{1,n-1}^{(1)} &= a_n^{(0)} \oplus a_0^{(0)},
\end{aligned}
\tag{5.16}
$$

such that $a_{U,i}^{(1)} x^i$ $(0 \leq i \leq m-1)$ and $a_{U,i}^{(1)} x^i$ $(0 \leq i \leq n-1,\ n+1 \leq i \leq m)$ can be selected to constitute $A_1^{(0)}$ and $A_1^{(1)}$, respectively. We can similarly extend (5.15) to

$A_U^{(2)} = \sum_{i=0}^{m+1} a_{U,i}^{(2)} x^i = a_{U,0}^{(2)} + \cdots + a_{U,m+1}^{(2)} x^{m+1}$, where x^{m+1} is an extended polynomial basis and

$$\begin{aligned} &a_{U,0}^{(2)} + a_{U,1}^{(2)} x + \cdots + a_{U,m-1}^{(2)} x^{m-1} = \sum_{i=0}^{m-1} a_i^{(0)} x^i, \\ &a_{U,m}^{(2)} = a_{1,n-1}^{(1)} = a_n^{(0)} \oplus a_0^{(0)}, \\ &a_{U,m+1}^{(2)} = a_{1,n-1}^{(2)} = a_n^{(1)} \oplus a_0^{(1)} = a_{n+1}^{(0)} \oplus a_1^{(0)}. \end{aligned} \tag{5.17}$$

Thus, $a_{U,i}^{(2)} x^i$ $(0 \le i \le m-1)$, $a_{U,i}^{(2)} x^i$ $(0 \le i \le n-1,\ n+1 \le i \le m)$ and $a_{U,i}^{(2)} x^i$ $(0 \le i \le n-2,\ n+1 \le i \le m+1)$, respectively, can be chosen to construct $A_1^{(0)}$, $A_1^{(1)}$, and $A_1^{(2)}$. In conclusion, we can have

$$\begin{aligned} A_U^{(t)} &= \sum_{i=0}^{m+t-1} a_{U,i}^{(t)} x^i \\ &= a_{U,0}^{(t)} + \cdots + a_{U,m+t-1}^{(t)} x^{m+t-1}, \end{aligned} \tag{5.18}$$

where $x^{m+1}, \ldots, x^{m+t-1}$ are defined as extended polynomial basis and (applicable to two trinomials recommended by NIST, where $m - n > t$)

$$\begin{aligned} &a_{U,0}^{(t)} + a_{U,1}^{(t)} x + \cdots + a_{U,m-1}^{(t)} x^{m-1} = \sum_{i=0}^{m-1} a_{1,i}^{(0)} x^i, \\ &a_{U,m+j-1}^{(t)} = a_{1,n+j}^{(0)} \oplus a_{1,j-1}^{(0)} \ (1 \le j \le n+1), \\ &\cdots \quad \cdots \quad \cdots \\ &a_{U,m+t-n-1+j}^{(t)} = a_{1,n+j}^{(0)} \oplus a_{1,j}^{(0)} \oplus a_{1,2n+j+1}^{(0)} \\ &\qquad\qquad (1 \le j \le t-n-1), \end{aligned} \tag{5.19}$$

where $a_{U,i}^{(2)} x^i$ $(0 \le i \le m-1)$, $a_{U,i}^{(2)} x^i$ $(0 \le i \le n-1,\ n+1 \le i \le m)$, ..., $a_{U,i}^{(2)} x^i$ $(0 \le i \le n-t,\ n+1 \le i \le m+t-1)$ can be chosen, respectively, to construct $A_1^{(0)}$, $A_1^{(1)}$, ..., and $A_1^{(t)}$, i.e.,

$$\begin{aligned} &\xi(A_U^{(t)}, 0) = A_1^{(0)}, \\ &\cdots \quad \cdots \quad \cdots \\ &\xi(A_U^{(t)}, t) = A_1^{(t)}, \end{aligned} \tag{5.20}$$

where $\xi(\cdot)$ represents the bit-selection operation. Similarly, for C_2, we have

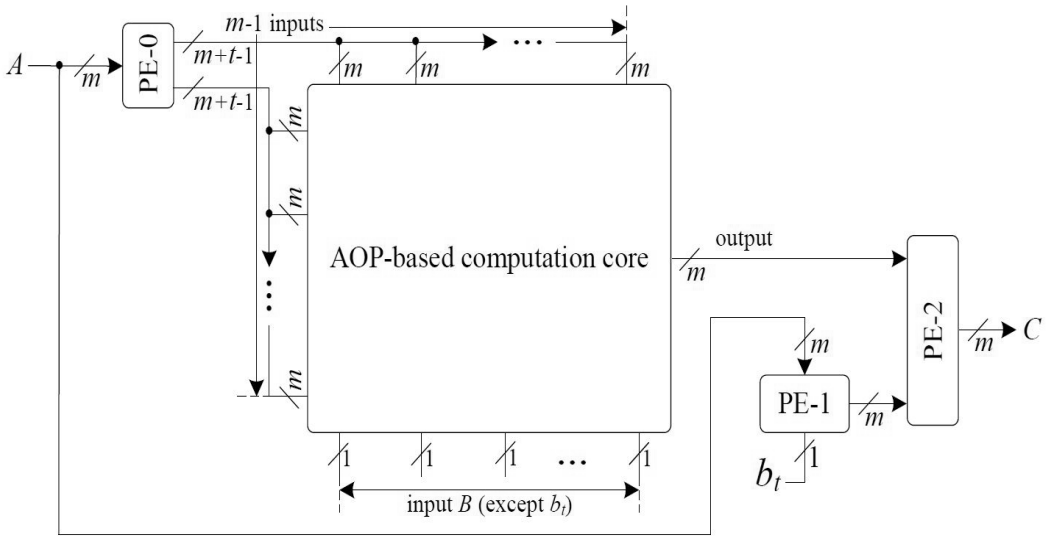

Figure 5.1: Proposed low register-complexity systolic multiplier based on the AOP-based computation core (MS-I), where the black box denotes the registers. (a) Proposed structure.

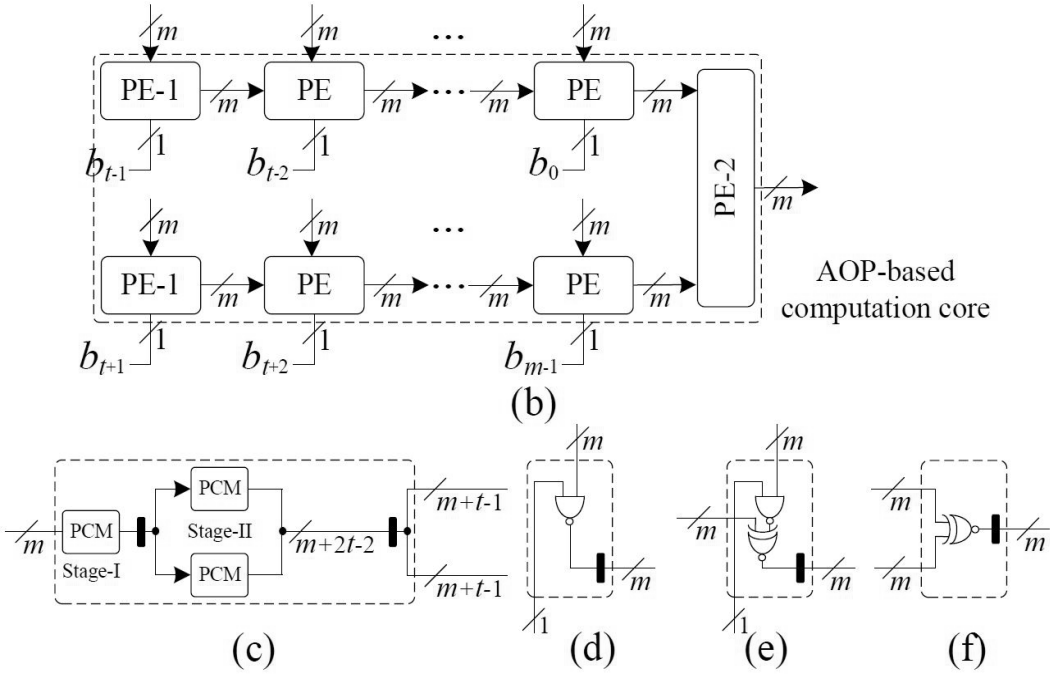

Figure 5.2: Proposed low register-complexity systolic multiplier based on the AOP-based computation core (MS-I), where the black box denotes the registers (b) Internal structure of the AOP-based computation core (MS-I, where $e = 2$). (c) Detailed design of PE-0. (d) Detailed design of PE-1. (e) Detailed design of regular PE. (f) Detailed design of PE-2..

$$\begin{aligned} \xi(A_V^{(t)}, 0) &= A_2^{(0)}, \\ \cdots \quad \cdots \quad \cdots & \\ \xi(A_V^{(t)}, t) &= A_2^{(t)}, \end{aligned} \tag{5.21}$$

where

$$\begin{aligned} A_V^{(t)} &= \sum_{i=0}^{m+t-1} a_{V,i}^{(t)} x^i \\ &= a_{V,0}^{(t)} + \cdots + a_{V,m+t-1}^{(t)} x^{m+t-1}, \end{aligned} \tag{5.22}$$

and (applicable to two trinomials recommended by NIST, where $m - n > t$)

$$\begin{aligned} a_{V,0}^{(t)} + a_{V,1}^{(t)} x + \cdots + a_{V,m-1}^{(t)} x^{m-1} &= \sum_{i=0}^{m-1} a_{2,i}^{(0)} x^i, \\ a_{V,m+j-1}^{(t)} &= a_{2,n-j}^{(0)} \oplus a_{2,m-j}^{(0)} \ (1 \le j \le t). \end{aligned} \tag{5.23}$$

Based on the above, (5.6) can be rewritten as

$$\begin{aligned} C &= C_1 + C_2 \\ &= Ab_t + \sum_{i=1}^{t} (\xi(A_U^{(t)}, i) b_{t-i} + \xi(A_V^{(t)}, i) b_{i+t}). \end{aligned} \tag{5.24}$$

From (5.15)-(5.16) and (5.18)-(5.19), we can derive $A_U^{(t)}$ and $A_V^{(t)}$ directly from operand A through XOR operations, and thus we define this operation as "pre-computed-modular" (PCM) operation as

$$\begin{aligned} A_U^{(t)} &= \mathrm{PCM}(A, U), \\ A_V^{(t)} &= \mathrm{PCM}(A, V). \end{aligned} \tag{5.25}$$

Based on (5.15)-(5.25), the proposed Montgomery multiplication algorithm for employing the AOP-based computation core is thus given by Algorithm 1:

Algorithm 1 Proposed Montgomery multiplication algorithm

Inputs: A and B are the pair of polynomials in $GF(2^m)$ to be multiplied.

Output: $C = A \cdot B \ r^{-1} \mathrm{mod} \ f(x)$.

1. Initialization step

1.1 $r^{-1} = x^{-t}$.

1.2 $D = 0$, $E = 0$.

2. Multiplication step

2.1-a. $A_U^{(t)} = \mathrm{PCM}(A, U)$.

2.1-b. $A_V^{(t)} = \mathrm{PCM}(A, V)$.

2.1-c. $D = Ab_t$.

2.2. $E = D + \sum_{i=1}^{t} (\xi(A_U^{(t)}, i)b_{t-i} + \xi(A_V^{(t)}, i)b_{i+t})$.

3. Final step

3.1. $C = E$.

---

where Step 2.2 refers to the bit-parallel multiplication process. According to the proposed algorithm, we generate operands at the first cycle period, and then they are distributed into t partial products to be accumulated in a systolic way, which greatly facilitates employing of the proposed AOP-based computation core since all involved bits are already generated from PCM (the details can be seen in the following subsection).

5.3 Proposed Low Register-Complexity Systolic Structure Employing the AOP-based Computation Core

The proposed structure based on the proposed Algorithm 1 (employing the proposed AOP-based computation core) is shown in Fig. 5.1(a). It contains one AOP-based computation core and three extra PEs. PE-0 yields two outputs (each output with $m + t - 1$ bits) to the computation core to be selectively connected with $m - 1$ input ports. As shown in Fig. 5.2(c), PE-0 performs the PCM of operand A and yields two outputs ($A_U^{(t)}$ and $A_V^{(t)}$) to the computation core, respectively (m bits of operand A are shared). The PCM of (31) only takes one XOR delay while the PCM of (27) takes two XORs' propagation time. To lower the critical-path delay, we have used two stage XOR operations to minimize the critical-path delay to one XOR delay (stage-I uses the least number of XOR gates required by (27), while stage-II realizes the rest of operations of (27) and (31)), as shown by an example design in Fig. 5.3. PE-1 calculates the multiplication of operand A and b_t according to Algorithm 1, while PE-2 functions as the final addition to produce the output C.

The internal structure of AOP-based computation core is shown in Fig. 5.2(b), where we

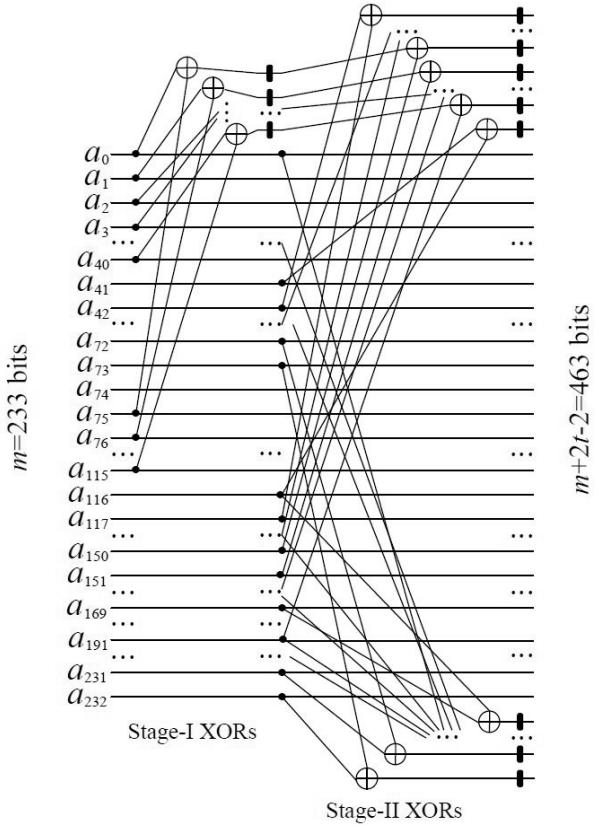

Figure 5.3: Detailed design of two stage XOR operations in PE-0 for trinomial $f(x) = x^{233} + x^{74} + 1$, where the black box denotes bit-register.

have used PEs from MS-I as internal PEs for $e = 2$ (one can extend the structure to any value of e). The computation core contains $(2t + 1)$ PEs, where the detailed designs of PEs are shown in Fig. 5.2(d)-(f), respectively. PE-1 performs a multiplication between one m bits operand and one bit of operand B and then yield the result to their right. The regular PE performs a multiplication between selected operand and one bit of operand B. The result of multiplication is added with the input from previous PE and then produces the result to the PE on its right. The last PE, PE-2, performs the addition of two systolic arrays and yields the final result.

The duration of maximum cycle period of the proposed multiplier of Fig. 5.1 is $(T_{NA} + T_{XN})$ (if we choose the PEs from MS-II, the critical-path delay will be T_{XN}). The proposed design gives the first output of desired product $(t + 3)$ cycles after the pair of operands are fed to the structure, while the successive outputs are produced in every cycle thereafter.

Table 5.1: COMPARISON OF AREA-TIME COMPLEXITIES OF VARIOUS SYSTOLIC MULTIPLIERS BASED ON TRINOMIALS

Design	AND	NAND	XOR	XNOR	Register	Latency	Critical-path delay
Bit-parallel systolic structures							
[7]	m^2	0	m^2+m-1	0	$4m^2+2m-2$	$2m-1$	T_A+T_X
[8][1]	0	m^2	m^2-1	0	$2m^2-2m$	m	$T_{NA}+T_X$
[8][2]	0	m^2	$m^2+m-2\sqrt{m}$	0	$2m^2+m\sqrt{m}-2m$	$2\sqrt{m}$	$T_{NA}+T_X$
Fig. 4.4 [10]	0	m^2	$< 1.5m^2+0.5m+1$	0	$1.5m^2+0.5m$	$m+1$	$2T_X$
Fig. 4.5 [10]	0	m^2	$< 1.5m^2+0.5m+1$	0	$1.5m^2+2m$	$m+2$	$T_{NA}+T_X$
Fig. 5.2[3*]	0	m^2	0	m^2-1	m^2+3m-1	$(m+7)/2$	$T_{NA}+T_{XN}$
Fig. 5.2[3*]	0	m^2	0	m^2-1	$2m^2+m$	$(m+9)/2$	T_{XN}
Fig. 10[4*]	0	m^2	0	m^2-1	$\simeq m^2+3m-1$	$(m-1)/e+3+\log_2 e$	$T_{NA}+T_{XN}$
Digit-parallel systolic structures ($d=2$)							
[8][5]	0	m^2	m^2+m	0	m^2	$m/2$	$T_{NA}+2T_X$
[9][6]	0	m^2	$\simeq m^2+m$	0	$\simeq 1.5m^2+2m$	$< 2\sqrt{m}$	$T_{NA}+T_X$
Fig. 8[7*]	0	m^2	0	m^2-1	$m^2/2+m-1/2$	$(m+11)/4$	$T_{NA}+2T_{XN}$
Fig. 8[7*]	0	m^2	0	m^2-1	$m^2-m+m-1/2$	$(m+11)/4$	$2T_{XN}$
Fig. 5.4[8*]	0	m^2	0	m^2-1	$\simeq m^2/2+1.5m$	$(m-1)/(2e)+2+\log_2(e/2)$	$T_{NA}+2T_{XN}$

*: The XOR gates in the PE-0 have been counted as XNOR gates.
1: The regular systolic structure.
2: Super-systolic structure.
3: Structure with $e=2$ and $d=1$ (PEs of the computation core are from MS-I).
4: Structure with $d=1$ (MS-I) (PEs of the computation core are from MS-I).
5: Digit-level structure for $d=2$.
6: The structure here can be seen as digit-level structure of $d=2$.
7: Structure with $e=2$ and $d=2$.
8: Structure with $d=2$ (MS-I) (PEs of the computation core are from MS-I).

5.4 Low-Latency Structure

Let $2t = eh + l$, where $0 \le l \le h$. For simplicity, we can assume $l = 0$, however, it can be extended to $l \neq 0$. Then, we can rewrite (5.24) as

$$\begin{aligned} C = Ab_t &+ \sum_{i=1}^{h} (\xi(A_U^{(t)}, i)b_{t-i} + \xi(A_V^{(t)}, i)b_{i+t}) \\ &+ \sum_{i=h+1}^{2h} (\xi(A_U^{(t)}, i)b_{t-i} + \xi(A_V^{(t)}, i)b_{i+t}) \\ &+ \ldots + \ldots + \ldots \\ &+ \sum_{i=t-h+1}^{t} (\xi(A_U^{(t)}, i)b_{t-i} + \xi(A_V^{(t)}, i)b_{i+t}). \end{aligned} \tag{5.26}$$

where the original two systolic arrays in the computation core of Fig. 5.2 can be divided into e arrays, as shown in Fig. 5.4. The latency of the structure in Fig. 5.4 is only $(h + 3 + \log_2 e)$ cycles, which is significantly shorter than the previous one in Fig. 4.2. A pipelined adder tree is used to add together the results of e systolic arrays of the computation core.

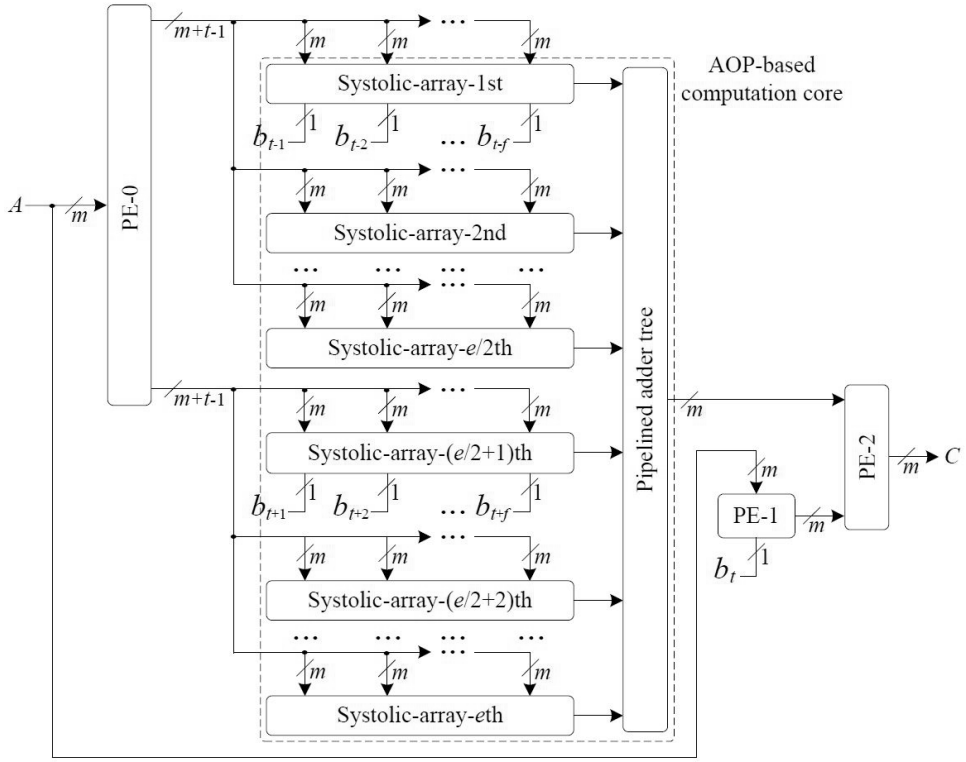

Figure 5.4: Proposed low-latency systolic multiplier.

5.5 Digit-Parallel Structure

We can also employ the PEs from Fig. 4.8 to have digit-parallel structure to reduce the register-complexity further. It is noted that digit-parallel structure can be combined with low-latency one to achieve optimal implementation.

5.6 Area and Time Complexities

5.6.1 Comparison

The area and time complexities in terms of logic gate count, register count, latency, and critical-path of the proposed structures and existing structures of [7-10] are listed in Table 5.1.

The proposed designs outperform the existing designs, especially in the register count. The proposed designs have lower area-time complexity than the design of [7]. When compared with the low-latency super-systolic structure of [8], the proposed design (Fig. 5.4) has shorter latency (if we choose $e = \sqrt{m}$) and less registers. Furthermore, when compared to the two recent designs in [9] and [10], the proposed designs not only have

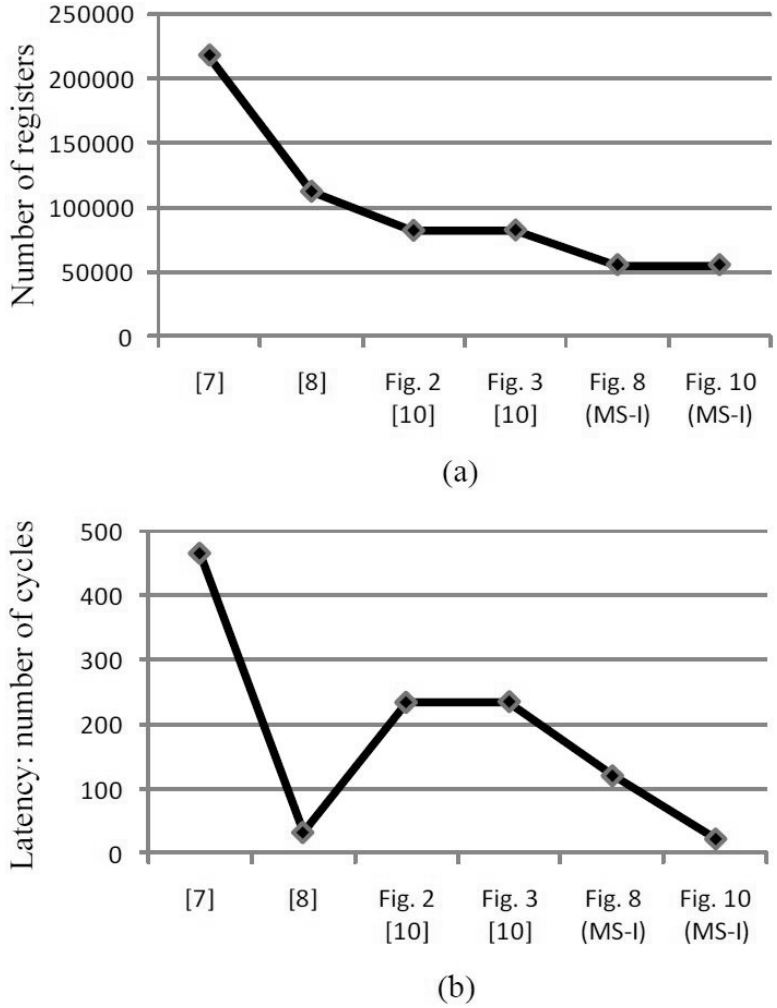

Figure 5.5: Comparison of register count and latency of various bit-parallel structures based on trinomial $f(x) = x^{233} + x^{74} + 1$ ([8] refers to the super-systolic structure). (a) Comparison of number of registers required by various designs. (b) Comparison of latency (number of cycles) for various designs (we have chosen $e = 16$ for the proposed structure of Fig. 10).

lower register count, but also involve significantly lower latency. Among all the existing designs, only the authors of [8-9] proposed the similar digit-parallel structures as Fig. 4.8. From Table III, it is shown that the proposed digit-parallel structures have less register-count and shorter latency than those of [8-9].

For a fair comparison, we have also given the comparison of register count and latency of various designs based on trinomial $f(x) = x^{233} + x^{74} + 1$, as shown in Fig 11. It can be seen that the proposed design, especially Fig. 8 (MS-I) has the best performance in both register count and latency.

5.6.2 FPGA Implementations

We have implemented the proposed designs, including the structures of Figs. 5.1 and 5.2 ($e = 2$) and Fig. 10 ($e = 16$, $d = 2$), using Xilinx ISE 14.1 on the Virtex 6 family device based on the trinomial $f(x) = x^{233} + x^{74} + 1$. The area-time-power complexities of the best existing designs ([9] and Fig. 3 of [10]) are also estimated. The obtained area-time-power complexities of all these designs are shown in Table IV.

As shown in Table IV, the proposed structures significantly outperform the existing designs. The proposed structures are found to have at least 70.0% and 47.6% less ADP and PDP than the corresponding designs, respectively.

5.6.3 Discussion

It is noted that from Table IV, the proposed design of Fig. 5.4 ($e = 16$ and $d = 2$) achieves the best area-time complexity among all the designs. The reduction of registers brought by digit-parallel implementation is significant. For practical applications, one can choose suitable value of d (coordinating with the selection of e) to obtain optimal realization.

5.7 Conclusion

A novel strategy for low complexity implementation of finite field multipliers over $GF(2^m)$ based on trinomials on FPGA platform has been proposed. We have proposed a modified data broadcasting technique to reduce the register-complexity within existing AOP-based multipliers. Then, the AOP-based multipliers are packed as standard computation cores to be used for trinomial based multipliers. A novel low register-complexity Montgomery multiplication algorithm for systolic trinomial-based finite field multipliers is presented. The systolic multiplier based on the proposed algorithm can employ the AOP-based computation core to offer low register-complexity implementation. We have also introduced structures for low-latency and digit-parallel implementations. Both the theoretical analysis and synthesis results have confirmed the higher efficiency of the proposed designs than the competing designs.

Table 5.2: COMPARISON OF AREA-TIME COMPLEXITIES OF VARIOUS DESIGNS BASED ON TRINOMIAL $f(x) = x^{233} + x^{74} + 1$

Design	Area	Delay[1]	Power	ADP[2]	PDP[3]
Bit-parallel systolic structures					
Fig. 4.5 ([10])	81,805	222.1	2.515	18,168,891	558.58
Figs. 5.1 and 5.2[4]	54,032	128.2	2.336	6,926,902	299.48
Fig. 5.4[5]	56,400	37.29	2.351	2,103,156	87.67
Digit-parallel systolic structures ($d = 2$)					
[9][6]	81,911	35.60	2.516	2,916,032	89.57
Fig. 5.4[7]	22,160	22.04	2.130	488,406	46.95

Unit for Area: number of slice register; Unit for delay: ns; Unit for power: W (power is estimated at 100MHz).
[1]: Delay = Latency.
[2]: ADP: Area-delay product = Area×Delay.
[3]: PDP: Power-delay product = Power×Delay.
[4]: based on structure of MS-I with $e = 2$.
[5]: based on structure of MS-I with $e = 16$ and $d = 1$.
[6]: structure here has 16 parallel systolic arrays ($d = 2$).
[7]: based on structure of MS-I with $e = 16$ and $d = 2$.

Chapter 6

Conclusion

This design focuses on finite field trinomial multiplier with AOP core, and also use Montgomery algorithm to improve the speed of the multiplier. The speed and complexity are two main points of hardware. There are multiple structures and algorithm which had been developed depending on the previous designs. In this thesis, our design focus on finite field which can reduce the complexity of arithmetic.

6.1 Low complexity multiplier based on trinomial

We present a low complexity multiplier in the beginning. This multiplier is based on AOP. There are several papers, which shown the structures of AOP multiplier. But depending on the signal flow diagram with two different ways of cut-set, we observed the registers of each PE can be saved. And according to the structures of AND, NAND, XOR and XNOR, we know the NAND and XNOR are faster than AND and XOR. Thus we change all AND and XOR with NAND and XNOR. For this new structure, we significantly decrease the area and latency.

6.2 Efficient systolic structure trinomial multiplier over $GF(2^m)$

For classic trinomial usually need to apply large amount of XOR gate and registers. But according to the structures, we know for each PE there just has one XOR gate difference. So we can design an independent preoccupation component which consist 44

by all XOR gates. In this way, we can combine preoccupation component with AOP multiplier together to build a new trinomial multipliers. Apart from changing structure, we can use another algorithm called Montgomery algorithm. Using this algorithm, the multiplier can reduce the latency but keep the same critical path.

6.3 Digital-Parallel Systolic structure trinomial multiplier over $GF(2^m)$

After using Montgomery algorithm to reduce the latency. We still can use another strategy to inproce the speed, that is using digital-parallel way to pipeline PEs. We assume there has m PEs, before we connect all PEs in one line, but now we separate one line systolic structure in several lines. In this way, all systolic structure line can do calculation together. Thus, the latency can be decreased. In all, our design are finished by above three steps, which can save the source and decrease the latency of multiplier

Chapter 7

Future Research

7.1 How to improve this multiplier with different structures

After finishing this thesis, we know that if we want to design a better circuit, we need to combine using new algorithm and optimizing structure. Only try to decrease the components based on the structure without applying new algorithm has its limitation. Montgomery algorithm act an important role this thesis. Similar strategy can be used in some new algorithms, such as Karatsnba algorithm or TMVP. Using different algorithm can get totally different structures which can change the latency and area significantly.

7.2 Use the same algorithm to different circuits

In this thesis, we focus on trinomial, so we still can use similar structure to implement different function, such as pentonomial.

Chapter 8

Publication

P. Chen, S. N. Basha, M. M.-Kermani, R. Azarderakhsh, and J. Xie, "FPGA Realization of Low Register Systolic All-One-Polynomial Multipliers over $GF(2^m)$ and Their Applications in Trinomial Multipliers,"IEEE Trans. VLSI Systems, submitted for review.

Chapter 9

Reference

[1] I. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*, ser. London Mathematical Society Lecture Note Series. Cambridge, U.K.: Cambridge Univ. Press, 1999.

[2] N. R. Murthy, and M. N. S. Swamy, "Cryptographic applications of Brahmaqupta-Bha skara equation," *IEEE Trans. on Circuits and Systems-I*, vol. 53, no. 7, pp. 1565-1571, 2006.

[3] M. Sun, L. E. Burke, Z.-H. Mao, Y. Chen, H.-C. Chen, Y. Bai, Y. Li, C. Li, and W. Jia. "eButton: A wearable computer for health monitoring and personal assistance," *Annual Design Automation Conference (DAC '14).* ACM, New York, NY, USA, Article 16.

[4]Y. Bai, C. Li, W. Jia, J. Li, Z.-H. Mao, and M. Sun "Designing a wearable computer for lifestyle evaluation," *in Proc. Annual Northeast Bioengineering Conference*, pp. 93-94, 2012 March 16-18; Philadelphia, PA.

[5] National Institute of Standards and Technology, "FIPS 186-2, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-2," 2000.

[6] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation.* New York: Wiley, 1999.

[7] C.-Y. Lee, J.-S. Horng, I.-C. Jou, and E.-H. Lu, "Low-complexity bit-parallel systolic montgomery multipliers for special classes of $GF(2^m)$," *IEEE Trans. Comput.*, vol. 54, no. 9, pp. 1061-1070, 2005.

[8] P. K. Meher, "Systolic and super-systolic multipliers for finite field $GF(2^m)$ based on irreducible trinomials," *IEEE Trans. Circuits and Systems-I*, vol. 55, no. 4, pp. 1031-1040, May, 2008.

[9] J. Xie, P. K. Meher and J. He, "Low-latency area-delay-efficient systolic multiplier over $GF(2^m)$ for a wider class of trinomials using parallel register sharing," *in Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)-2012*, pp. 89-92, 2012.

[10] S. B.-Sarmadi, and M. Farmani, "High-throughput low-complexity systolic Montgomery multiplication over $GF(2^m)$ based on trinomials," *IEEE Trans. on Circuits and Systems-II*, vol. 62, no. 4, pp. 377-381, 2015.

[11] J. Xie, J. He and P. K. Meher, "Low latency systolic Montgomery multiplier for finite field $GF(2^m)$ based on pentanomials," *IEEE Trans. on VLSI Systems*, vo. 21, no. 2, pp. 385-389, 2013.

[12] P. Montgomery, "Modular multiplication without trial division," *Math, Computation*, vol. 44, no. 170, pp. 519-521, 1985.

[13] R. Azarderakhsh, K. Jarvinen, and V. Dimitrov, "Fast inversion in $GF(2^m)$ with normal basis using hybrid-double multipliers," *IEEE Trans. on Comput.*, vol. 63, no. 4, pp. 1041-1047, 2014.

[14] R. Azarderakhsh, D. Jao, and H. Lee, "Space complexity reduction algorithms for Gaussian normal basis multiplication," *IEEE Trans. on Information Theory*, vol. 61, no. 5, pp. 2357-2369, 2015.

[15] R. Azarderakhsh, M. Mozaffari-Kermani, "High-performance two-dimensional finite field multiplication and exponentiation for cryptographic applications," *IEEE Trans. Computer Aided Design Integrated Circuits Systems*, vol. 34, no. 10, pp. 1-8, 2015

[16] C-Y. Lee and P. K. Meher, "Area-efficient subquadratic space-complexity digit-serial multiplier for type-II optimal normal basis of $GF(2^m)$ using symmetric TMVP and block recombination techniques," *IEEE Trans. on Circuits and Systems-I*, vol. 62, no. 12, pp. 2846-2855, 2015.

[17] S. Talapatra, H. Rahaman, and S. K. Saha, "Unified digit serial systolic Montgomery multiplication architecture for special classes of polynomials over $GF(2^m)$," *in Conf. on Digital System Design: Architectures, Methods and Tools*, pp. 427-432, 2010.

[18] S. Fenn, and M. Parker, "Bit-serial multiplication in $GF(2^m)$ using all-one polynomials," *IEE proc. Com. Digit. Tech.*, vol. 144, no. 6, pp. 391-393, 1997.

[19] K.-Y. Chang, D. Hong, and H.S. Cho, "Low complexity bit-parallel multiplier for $GF(2^m)$ defined by all-one polynomials using redundant representation," *IEEE Trans. Comput.*, vol. 54, no. 12, pp. 1628-1629, 2005.

[20] H.-S. Kim, and S.-W. Lee, "LFSR multipliers over $GF(2^m)$ defined by all-one polynomial," *Integr., the VLSI jour.*, vol. 40, no. 4, pp. 571-578, 2007

[21] P. K. Meher, and C.Y. Lee, "An optimized design of serial-parallel finite field multiplier for $GF(2^m)$ based on all-one polynomials," *ASP-DAC 2009*, pp. 210-215, 2009.

[22] M.-Sandoval, M. F.-Uribe, and C. Kitsos, "Bit-serial and digit-serial $GF(2^m)$ Montgomery multipliers using linear feedback shift registers," *IET Comput. & Digital Tech.*, vol. 5, no. 2, pp. 86-94, 2011.

[23] C.-Y. Lee, E.-H. Lu, and J.-Y. Lee, "Bit-parallel systolic multipliers for $GF(2^m)$ fields defined by all-one and equally spaced polynomials," *IEEE Trans. Comput.*, vol. 50, no. 6, pp. 385-393, May, 2001.

[24] J. Xie, P. K. Meher and J. He, "Low-complexity multiplier for $GF(2^m)$ based on all one polynomials," *IEEE Trans. on VLSI Systems*, vol. 21, no. 1, pp. 168-172, 2013.

[25] Y.-R. Ting, E.-H. Lu, and Y.-C. Lu, "Ringed bit-parallel systolic multipliers over a class of fields $GF(2^m)$," *Integration, the VLSI journal*, vol. 38, no. 4, pp. 571-578, 2005

[26] C.-Y. Lee, J.-S. Horng, I.-C. Jou, and E.-H. Lu, "Low-complexity bit-parallel systolic montgomery multipliers for special classes of $GF(2^m)$," *IEEE Trans. Comput.*, vol. 54, no. 9, pp. 1061-1070, Sep. 2005.

[27] S. Talapatra, H. Rahaman, and J. Mathew, "Low complexity digit serial systolic Montgomery multipliers for special class of $GF(2^m)$," *IEEE Trans. VLSI Sys.*, vol. 18, no. 5, pp. 847-852, May, 2010.

[28] T. Itoh and S. Tsujii, "Structure of parallel multipliers for a class of fields $GF(2^m)$," *Information and Computation*, vol. 83, no.1, pp. 21-40, 1989.

[29] S.-B. Wicker, and V.K. Bhargava, *Reed − Solomon Codes and Their Applications*, Piscatawy, NJ: IEEE Press, 1994.

[30] J.-P. Deschamps, J.L.Imana, and G.D. Sutter, *Hardware implementation of finite field arithmetic.* McGraw-Hill, 2009.

[31] I. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography, ser.* London Mathematical Society Lecture Note Series. Cambridge, U.K.: Cambridge Univ. Press, 1999.

[32] K. K. Parhi, *VLSI Digital Signal Processing Systems : Design and Implementation.* New York: Wiley, 1999.

[33] S. Y. Kung, *VLSI array processors.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

[34] H. Fan, and M.A. Hasan, "Relationship between $GF(2^m)$ Montgomery and shifted polynomial basis multiplication algorithms," *IEEE Trans. Computers*, vol. 55, no. 9, pp. 1202-1206, 2006.

[35] C.-S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite Fields $GF(2^m)$," *IEEE Trans. Comput.*, vol. C-33, no. 4, pp. 357C360, Apr. 1984.

[36] Digital Signature Standard (DSS), FIPS 186-2, *National Institute of Standards and Technology*, 2000.

[37] P. K. Meher, "Systolic and non-systolic scalable modular designs of finite field multipliers for Reed-Solomon Codec," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 6, pp. 747C757, June, 2009.

[38] S. K. Jain, L. Song, and K. K. Parhi, "Efficient semisystolic architectures for finite field arithmetic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* vol. 6, no. 1, pp. 734-749, Mar. 1998.

[39] C.-S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite fields $GF(2^m)$," *IEEE Trans. Comput.*, vol. C-33, no. 4, pp. 357C360,Apr. 1984.

[40] C.-L. Wang and J.-L. Lin, "Systolic array implementation of multipliers for finite fields $GF(2^m)$," *IEEE Trans. Circuits Syst.*, vol. 38, no. 7, pp. 796C800, Jul. 1991.

[41] C. H. Kim, C. P. Hong, and S. Kwon, "A digit-serial multiplier for finite field $GF(2^m)$," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 4, pp. 476-483, 2005.

[42] N-Y. Kim, H-S. Kim, and K-Y. Yoo, "Computation of AB^2 multiplication in $GF(2^m)$ using a low-complexity systolic architecture," *IEE Proc.−Circuits Devices Syst.*, vol. 150, no. 2, pp. 119-123, April, 2003.

[43] L. Song, K. K. Parhi, I. Kuroda, and T. Nishitani, "Hardware/software codesign of finite field datapath for low-energy Reed-Solomn codecs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* vol. 8, no. 2, pp. 160-172, Apr. 2000.

[44] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore Curve-Based Cryptoprocessor with Reconfigurable Modular Arithmetic Logic Units over $GF(2^n)$," *IEEE Trans. Comput.* Vol. 56, no. 9, pp. 1269-1282, 2007.

[45] H. Wu, "Bit-parallel polynomial basis multiplier for new classes of finite fields," *IEEE Trans. Computers*, vol. 57, no. 8, pp. 1023-1031, 2008.

[46] C. Paar, "Low complexity parallel multipliers for Galois fields $GF((2^n)4)$ based on

special types of primitive polynomials", $IEEE\ International\ Symposium\ on\ Information\ Theory$, 1994.

[47] P. K. Meher, "On efficient implementation of accumulation in finite field over $GF(2^m)$ and its applications," $IEEE\ Trans.\ Very\ Large\ Scale\ Integr.\ (VLSI)\ Syst.$, vol. 17, no. 4, pp. 541-550, 2009.

[48] T.-C. Chen, S.-W. Wei, and H.-J. Tsai, "Arithmetic unit for finite field $GF(2^m)$," $IEEE\ Circuits\ and\ System-I$, vol. 55, no. 3, pp. 828-837, 2008.

[49] N. R. Murthy, and M. N. S. Swamy, "Cryptographic applications of brahmaquptabha skara equation,"

$IEEE\ Circuits\ and\ System-I$, vol. 53, no. 7, pp. 1565-1571, 2006.

[50] C. Spagnol, E. M. Popovici, and W. P. Marnane, "Hardware implementation of $GF(2^m)$ LDPC decoder," $IEEE\ Circuits\ and\ System-I$, vol. 56, no. 12, pp. 2609-2620, 2009.